Inteligenţă artificială

Versiunea 15 iunie 2020

Lucian M. Sasu, Ph.D.

Cuprins

Capitolul 1

Introducere

Inteligenţa computaţională (IC) este un domeniu care combină elemente de învăţare automată, adaptare, evoluţie şi logică fuzzy pentru a rezolva probleme care, abordate tradiţional, sunt dificil sau imposibil de abordat. Este o ramură a inteligenţei artificiale. Subdomeniile majore ale inteligenţei computaţionale sunt:

- modele de învăţare parametrică
- modele de învăţare neparametrice, precum reţele neurale[1] artificiale, Support Vector Machines, modele probabiliste;
- mulţimi şi logică fuzzy;
- calcul evoluţionist;
- imunitate artificială;
- inteligenţa muşuroiului.

Fiecare din aceste subdomenii a evoluat rapid şi s–au impus ca potenţiale metode de rezolvare efectivă a unor probleme complexe şi presante, pentru care abordările uzuale sunt nefructuase. De regulă, prototipizarea unui sistem inspirat din inteligenţa computaţională este rapidă, iar pentru o problemă se pot folosi mai multe abordări: de exemplu, optimizarea se poate face prin algoritmi genetici sau prin anumite familii de reţele neurale.

Metodele din inteligenţa computaţională sunt frecvent inspirate din biologie: reţelele neurale au pornit de la modelul imaginat pentru neuronul biologic, calculul evoluţionist este bazat pe teoria evoluţiei şi pe genetică. Sistemele fuzzy sunt introduse pentru a permite manipularea impreciziei, altfel decât prin teoria probabilităţilor.

Este o mare diferenţă între abordarea clasică, algoritmică a unei probleme şi cea dată de IC. În primul caz este pusă la bătaie toată abilitatea

[1]Sau “neuronale”

celui care imaginează algoritmul pentru a rezolva problema; este un demers anevoios, depinzând esenţial de imaginaţia, puterea de abstractizare şi experienţa persoanei în cauză; evident, este un proces creativ, la ora actuală efectuat exclusiv de oameni. Totodată, de cele mai multe ori rezultatele sunt exacte; de asemenea, se are în vedere permanent micşorarea complexităţii de calcul a problemei respective, dar în destule situaţii o soluţie exactă presupune un efort de calcul sau resurse de memorie prohibitive.

Abordarea IC este total diferită: pentru reţele neurale sau algoritmi genetici, definitorie este capacitatea de *adaptare* automată sau *auto-organizare* la condiţiile problemei. Este modelul inspirat din natură, unde un sistem biologic preia semnale din mediu şi printr-un proces de învăţare se adaptează, astfel încât să îşi îndeplinească scopul, sau pentru a obţine o mai bună integrare. Soluţia la care se ajunge nu este neapărat optimă, dar este un răspuns suficient de bun pentru problema propusă. În implementarea unui sistem din cadrul IC accentul cade mai mult pe abilitatea sistemului rezultat de a se adapta, de a învăţa, decât pe imaginaţia celui care îl concepe. Abilităţile de programare pentru implementarea sau personalizarea sistemului sunt însă esenţiale.

Sistemele propuse în cadrul IC sunt cu un mare grad de aplicabilitate. De exemplu, algoritmii genetici pot fi folosiţi pentru o clasă largă de funcţii, nedepinzând atât de mult precum cercetările operaţionale de ipoteze care în practică pot fi prea restrictive.

O definiţie a "inteligenţei" potrivită pentru contextul de IC este:

Definiţia 1. *Inteligenţa este abilitatea unui sistem de a–şi adapta comportamentul pentru a–şi îndeplini scopurile în mediul său. Este o proprietatate a tuturor entităţilor ce trebuie să ia decizii şi al căror comportament este condus de scop.*

Definiţia de mai sus a fost dată în 1995 de către David Fogel, scoţând în evidenţă elementul esenţial al comportamentului inteligent şi în particular al inteligenţei computaţionale: adaptarea.

Reţelele neurale artificiale reprezintă grupuri interconectate de neuroni artificiali care au abilitatea de a învăţa din şi a se adapta la mediul lor, construind un model al lumii. Ele au apărut ca răspuns la modelarea activităţii creierului biologic, precum şi ca modalitate propusă pentru a obţine sisteme artificiale capabile să recunoască şabloane. Exemple de reţele neurale şi algoritmi de instruire se găsesc în [1], [2], [3].

Sistemele fuzzy sunt introduse pentru a putea gestiona imprecizia, noţiunile vagi ("înalt", "acum") şi aproximarea. Sunt elemente des întâlnite în modelarea de limbaj sau în situaţii de cunoaştere incompletă. Teoria mulţimilor fuzzy permite ca un element să aibă un anumit grad de apartenenţă (număr între 0 şi 1) la o mulţime, spre deosebire de teoria clasică a mulţimilor. Logica fuzzy permite considerarea mai multor valori de adevăr decât cele din logica clasică, sau altfel zis, a unor grade de adevăr diferite. Este variantă de realizare a raţionamentului aproximativ.

Calculul evoluţionist se ocupă în special cu optimizarea şi de probleme de căutare, bazate pe mecanismele preluate din genetică şi evoluţionism. Se pleacă de la ideea evoluţiei unei populaţii de indivizi, fiecare din ei fiind o soluţie potenţială a problemei ce se vrea rezolvată. Domeniul include algoritmii genetici, programarea evoluţionistă, programarea genetică şi strategii de evoluţie.

Sistemele rezultate prin inteligenţă computaţională pot reprezenta hibridizări ale celor de mai sus; de exemplu, există sisteme neuro-fuzzy, iar ajustarea parametrilor pentru un sistem adaptiv se poate face prin algoritmi genetici. Alegerea uneltei potrivite pentru problema în cauză este o problemă deloc simplă, deoarece de regulă se pot folosi mai multe abordări.

1.1 Reţele neurale

1.1.1 Bazele biologice

Reţeaua neurală biologică a evoluat de-a lungul timpului, ajungând la performanţe care astăzi nu sunt accesibile calculatoarelor electronice: de exemplu, recunoaşterea de imagini, specifică animalelor; sau interpretarea ecoului reflectat de către obstacole sau insecte, în cazul liliecilor - chiar dacă au creierul foarte mic, procesarea în cazul lor se face mai rapid decât pentru cele mai performante sisteme electronice actuale.

Studiile efectuate în ultimul secol au permis enunţarea unor prinicipii asupra modului de funcţionare a sistemelor neurale biologice; suntem însă departe de a cunoaşte toate detaliile funcţionale şi structurale. Chiar şi aşa, prin implementarea modelelor obţinute, rezultatele sunt mai mult decât notabile.

Figura 1.1 ([4]) reprezintă cel mai comun tip de neuron natural. În scoarţa neurală există circa 86 de miliarde de neuroni interconectaţi, fiecare putând avea până la 10^4 conexiuni cu alţi neuroni; modul de grupare a acestora şi interdependenţele nu sunt pe deplin cunoscute.

Un neuron artificial are o structură asemănătoare, fiind un element de procesare conectat cu alte elemente ce preia intrare de la nişte neuroni şi produce o ieşire ce devine intrare pentru alţi neuroni; legăturile neurale sunt nişte coeficienţi numerici, iar prin algoritmi de învăţare se obţine adaptarea convenabilă a reţelei neurale. Adaptarea (sau învăţarea) este aspectul esenţial al reţelelor neurale: plecând de la seturi de date, se detectează automat şabloanele existente şi se construiesc niste modele care pot fi folosite mai departe.

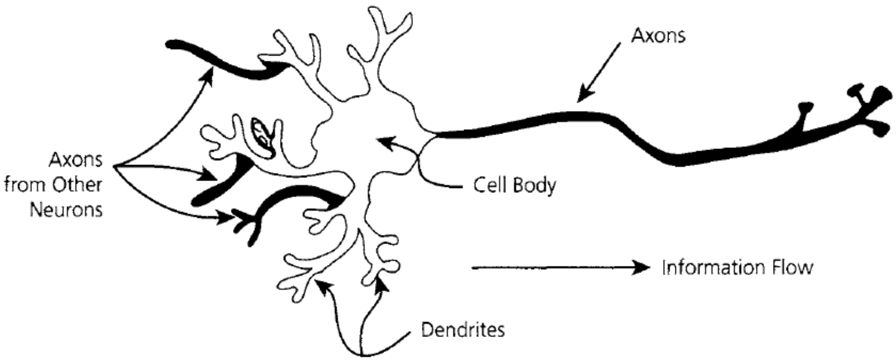


Figura 1.1: Neuron natural [4]

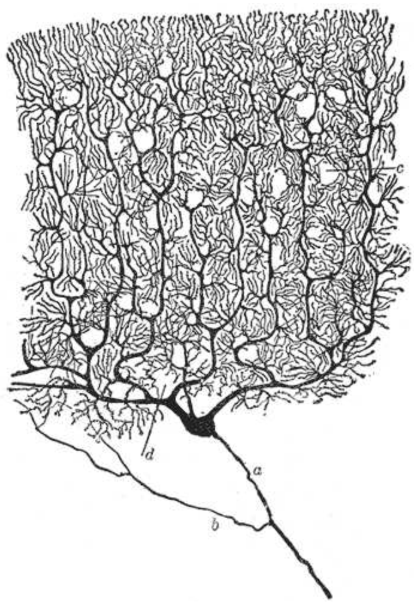


Figura 1.2: Neuron de tip Purkinje din cortexul cerebelar; sursa http://en.wikipedia.org/wiki/Neuron.

1.1.2 Diferenţe între RN artificiale şi naturale

În mod cert însă, există diferenţe: nu sunt modelate toate tipurile cunoscute de neuroni; apoi, o lege biologică spune că un neuron poate să excite sau să inhibe un neuron cu care este conectat; în modelarea de RN artificiale, o pondere de legătură este fie excitatoare, fie inhibitoare, dar forma ei este fixată după ce s–a făcut învăţarea.

O altă diferenţă (şi punct de critică pentru reţelele neurale artificiale) este faptul că modelarea semnalului făcută sub formă de valori continue este de negăsit în reţelele biologice; în RN biologice se folosesc de fapt trenuri de impulsuri care sunt transmise către neuroni, apărând variaţie în frecvenţa semnalului. Acest aspect a fost abordat relativ târziu, în cadrul reţelelor neurale cu pulsuri.

Viteza reţelelor neurale este iarăşi un loc în care apar diferenţe. Se estimează că neuronii naturali au cicli de timp între 10 şi 100 milisecunde; implementările de RN artificiale funcţionează pe procesoare de câtiva gigahertzi, deci cu un ciclu de mai puţin de o nanosecundă. Chiar şi aşa, reţelele neurale biologice sunt cu mult mai performante decât cele artificiale.

Altă diferenţă este că neuronii naturali sunt grupaţi în cantităţi mari, uneori de sute de milioane de unităţi. Se ajunge astfel la un grad de paralelism masiv, ce e încă atins în modelel neurale actuale.

1.1.3 Aplicabilitate

- *Clasificarea* - pe baza unui set de date de forma (intrare - ieşire asociată) se construieşte un sistem care detectează asocierile dintre datele de intrare şi etichetele ce le sunt asociate; etichetele - sau clasele - sunt dintr-o mulţime discretă, finită. Clasificarea se foloseşte pentru recunoaşterea automată a formelor, recunoaşterea vorbirii, diagnoză medicală şi altele.

- *Estimarea de probabilitate condiţionată* - similar cu clasificarea, dar se produce un sistem care estimează probabilitatea ca un obiect să aparţină unei clase, date fiind trăsăturile de intrare; de exemplu, dat fiind conţinutul unui mesaj de email care este probabilitatea ca să fie mail legitim sau spam;

- *Regresie* - asemănător cu clasificarea, dar ieşirile nu sunt dintr-o mulţime discretă şi finită, ci valori numerice continue;

- *Memorie asociativă*, sau memorie adresabilă prin conţinut - se poate regăsi o dată pe baza unei părţi a ei. Este un mecanism diferit de modul în care calculatoarele regăsesc informaţia - pe baza adreselor sau a unei căutări - dar apropiată de modul în care se face regăsirea elementelor reţinute de către o persoană.

- *Grupare automată (clustering)* - pe baza similarităţilor existente într–un set de date, se detectează grupările de date; elementele dintr-un grup sunt mai apropiate între ele decât de altele din alt grup;

- *Detectara automată de trăsături* – a acelor elemente care fac ca procesul de recunoaştere a unui obiect să fie mai bun decât dacă se folosesc cunoştinţe specifice domeniului;

- *Controlul sistemelor* - folosite pentru cazul în care un proces trebuie să fie ghidat pentru a se încadra în parametri; utilitatea reţelelor neurale provine din faptul că nu se presupune că există dependenţe liniare între acţiune şi reacţiune.

1.2 Calcul evoluţionist

Principalele paradigme[2] ale calculului evoluţionist sunt:

- algoritmii genetici - evoluţia unei populaţii de indivizi (cromozomi), folosind selecţia, încrucişarea şi mutaţia;

- programarea evoluţionistă - similar cu precedenta, dar fără a folosi încrucişarea; este văzută ca evoluţia de specii diferite, între care nu există hibridizări;

- strategiile de evoluţie - similari cu algoritmii genetici, dar se foloseşte recombinarea în loc de încrucişare şi deseori alte metode de mutaţie

- programarea genetică - metode evolutive aplicate programelor de calculator.

1.2.1 Bazele biologice

Domeniile de inspiraţie sunt genetica şi teoria evoluţionistă. Genetica tratează ereditatea, adică transmiterea trăsăturilor de la părinţi la urmaşi. Astfel, adaptarea obţinută în generaţiile anterioare este preluată de către urmaşi şi continuată. Codificarea caracteristicilor este dată de cromozomi. Noţiunile şi mecanismele sunt preluate din teoria eredităţii întemeiată de Gregor Mendel şi teoria evoluţionistă a lui Charles Darwin.

[2] "Paradigma este o construcţie mentală larg acceptată, care oferă unei comunităţi sau unei societăţi pe perioada îndelungată o bază pentru crearea unei identităţi de sine (a activităţii de cercetare de exemplu) şi astfel pentru rezolvarea unor probleme sau sarcini.", conform Wikipedia.

1.2.2 Cromozomi

Cromozomii sunt structuri din interiorul celulelor care menţin informaţia genetică. În cazul oamenilor, sunt 46 de cromozomi, jumătate moşteniţi de la tată şi jumătate de la mamă. Cromozomii sunt alcătuiţi din gene, fiecare fiind identificată prin locaţia pe care o ocupă şi prin funcţia asociată.

1.2.3 Diferenţe între cromozomii biologici şi cei artificiali

Cromozomii artificiali sunt reprezentări simplificate a celor biologici. În timp ce neuronii biologici sunt secvenţe de acizi nucleici, cromozomii artificiali sunt şiruri de cifre binare.

Cromozomii biologici care definesc organismele vii variază în lungime, chiar dacă de la un organism la altul din aceeaşi specie pentru un cromozom specific lungimea este constantă. În algoritmii genetici, lungimea este fixă.

La reproducerea indivizilor dintr-o populaţie naturală, jumătate din informaţia genetică este preluată de la tată şi jumătate de la mamă. În algoritmii genetici, procentul de combinaţie poate să difere.

1.2.4 Aplicabilitate

Principala arie de aplicare este optimizarea, pentru situaţiile în care căutarea soluţiei cere un timp îndelungat. Algoritmii genetici sunt folositi ca o metodă euristică; problemele abordate sunt din cele mai diverse — optimizarea unui plan de lucru sau circuit, balansarea încărcării, optimizarea ingredientelor, design automat, încărcarea containerelor, optimizarea structurilor moleculare, testarea mutaţiilor, optimizarea sistemelor de compresie, selectarea modelelor optime, găsirea defectelor hardware etc.

1.3 Sistemele fuzzy

Sistemele fuzzy şi logica fuzzy nu sunt de inspiraţie biologică, ci preluate din partea comportamentală umană. Este o modalitate de manipulare a incertitudinii, modelând imprecizia, caracterul vag, ambiguitatea. Este vorba de un alt tip de incertitudine decât cel modelat prin intermediul variabilelor aleatoare din cadrul teoriei probabilităţilor. Se foloseşte pentru modelarea impreciziei lingvistice ("Maria e înaltă", "Livrarea se face în aproximativ 3 ore"). Teoria mulţimilor fuzzy a fost dezvoltată de către Lotfi Zadeh începând cu anul 1965.

Un exemplu de raţionament fuzzy este:

```
IF temperature IS very cold THEN stop fan
IF temperature IS cold THEN turn down fan
IF temperature IS normal THEN maintain level
IF temperature IS hot THEN speed up fan
```

Toate variantele sunt evaluate şi în funcţie de rezultat se ajustează viteza ventilatorului. Modelarea conceptelor de "very cold", "normal" etc. se face prin mulţimi vagi.

Pornind de la acest curent, s-au dezvoltat următoarele: fuzzificare/defuzzificarea, sisteme de control fuzzy, jocuri fuzzy, matematică fuzzy, teoria măsurii fuzzy, căutare fuzzy.

Teoria fuzzy este folosită intens în sisteme ce presupun control: camere video, sisteme de frânare sau accelerare, sisteme de control al debitului şi presiunii etc. De asemenea, sistemele expert din domeniu medical, financiar, navigaţional, diagnoza mecanica etc. se folosesc masiv de suportul pentru imprecizie şi ambiguitate.

1.4 Tipuri de învăţare în inteligenţa computaţională

Învăţarea permite unui sistem să se adapteze la mediul în care operează; pe baza semnalelor provenite din exterior, sistemul inteligent îşi modifică parametrii pentru o îndeplinire cât mai bună a sarcinii propuse. Trebuie făcută distincţia între "învăţare" şi "memorare cu regăsire exactă" – această din urmă problemă este rezolvată de structuri şi baze de date.

Există trei tipuri principale de învăţare:

1. supervizată
2. nesupervizată
3. prin întărire

La acestea se adaugă şi învăţarea semi–supervizată.

1.4.1 Învăţarea supervizată

Se presupune că există un "profesor" care poate prezenta un set de date de instruire având forma (intrare — ieşire asociată), relevant, care este preluat de către sistem şi învăţat. Se foloseşte o funcţie de eroare, care măsoară cât de departe este răspunsul cerut faţă de cel furnizat de sistem; pe baza erorii se desfăşoară un proces de ajustare a valorilor din sistemul computaţional inteligent până când eroarea scade sub un anumit prag. Rezultatul final este obtinerea unui sistem ce poate să furnizeze o valoare de ieşire adecvată pentru o anumită valoare de intrare ce nu este prezentă în setul de instruire.

Exemple de sisteme ce folosesc instruirea supervizată: perceptronul, perceptronul multistrat, Fuzzy ARTMAP, reţelele cu activare radială.

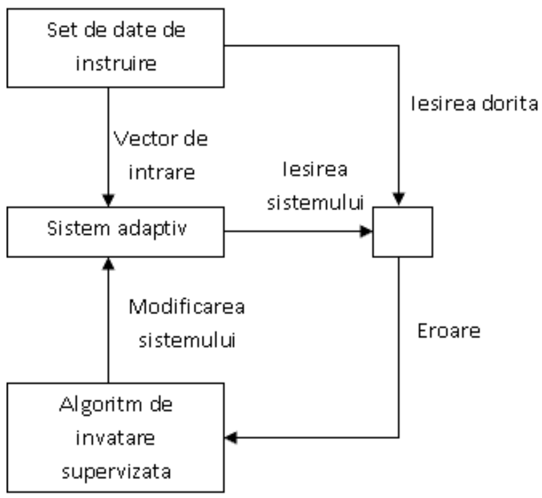


Figura 1.3: Schema de lucru pentru învăţare supervizată

1.4.2 Învăţarea prin întărire

Învăţarea prin întărire (eng: reinforcement learning) este similară cu învăţarea supervizată, numai că în loc de a se furniza ieşirea asociată unei intrări, se pune la dispoziţie o indicaţie care arată cât de bine a acţionat sistemul respectiv. Acesta este un sistem bazat pe critică sau aprobare, fiind instruit în raport cu măsura în care ieşirea obţinută de un sistem corespunde valorii dorite (dar fără ca această valoare dorită să fie precizată sistemului!). Rolul profesorului este luat de un critic, care precizează în ce măsură ieşirea obţinută se apropie de cea dorită. Pe termen lung, sistemul îşi va modifica propriul comportament astfel încât să se reducă criticile obţinute.

Acest tip de învăţare este plauzibil din punct de vedere biologic, deoarece o fiinţă sau un agent artificial inteligent va încerca să îşi minimizeze starea de disconfort prilejuită de comportament neadecvat. Rolul criticului este dat aici de mediul înconjurător. Schema de lucru este dată în figura 1.4.

1.4.3 Învăţarea nesupervizată

Spre deosebire de precedentele moduri de învăţare, în acest caz nu se primeşte niciun semnal de tip ieşire sau critică asociată. Sistemului capabil de grupare i se dau doar valori de intrare. El face o grupare automată sau foloseşte o învăţare de tip competititiv. Aplicatiile clasice sunt analiza asociererilor, gruparea pe baza de similaritate si estimarea de densitate de probabilitate.

Schema de lucru este dată în figura 1.5. Acest tip de adaptare este prezent în reţelele de tip clustering, analiza de asocieri, analiza componentelor principale etc.

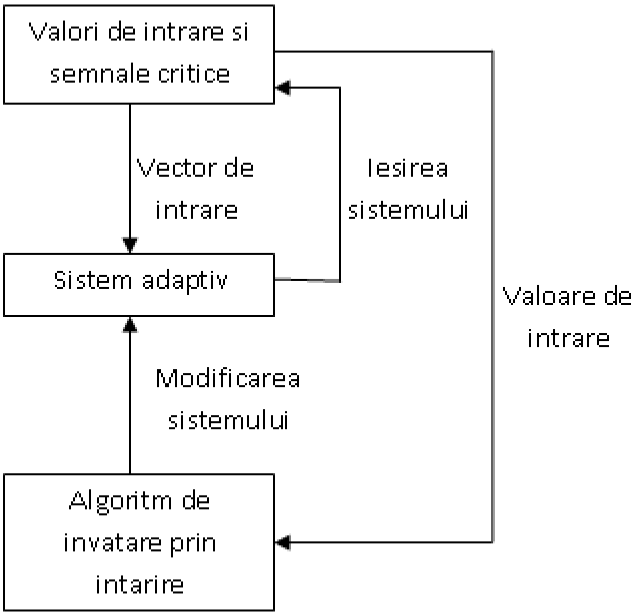


Figura 1.4: Schema de lucru pentru învăţare prin întărire

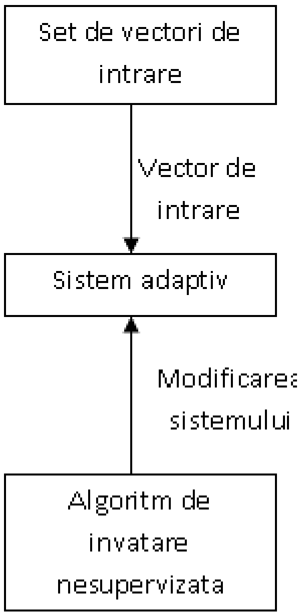


Figura 1.5: Schema de lucru pentru învăţare nesupervizată

1.5 Auto-organizarea

Auto-organizarea, alături de învăţare, este un alt atribut important al sistemelor computaţionale inteligente. Este prezentă în sistemele naturale, de exemplu în creierul nou născuţilor, unde auto-organizarea se manifestă în principal prin distrugerea legăturilor nefuncţionale. Auto-organizarea este definită astfel:

Definiţia 2. *Spunem că un sistem se auto-organizează dacă, după ce se primesc intrarea şi ieşirea unui fenomen necunoscut, sistemul se organizează singur astfel încât să simuleze fenomenul necunoscut [5].*

sau:

Definiţia 3. *Sistemele cu auto-organizare se auto-organizează pentru a clasifica percepţiile din mediu în percepţii ce pot fi recunoscute, sau şabloane [5].*

Capitolul 2

Regresia liniară

2.1 Exemplu şi notaţii

Notă: expunerea din acest curs este făcută după [6].

Regresia liniară este o metodă folosită pentru predicţia unei valori numerice dintr-o mulţime infinită de valori. Ca exemplu, să presupunem că vrem să facem predicţia costului unei proprietăţi imobiliare, dată fiind suprafaţa sa. Se cunosc date anterioare despre vânzarea unor astfel de proprietăţi. Pe baza acestor date vom construi o funcţie care să ne permită aproximarea preţului (număr real) pentru alte proprietăţi de interes. O exemplificare este dată în figura 2.1.

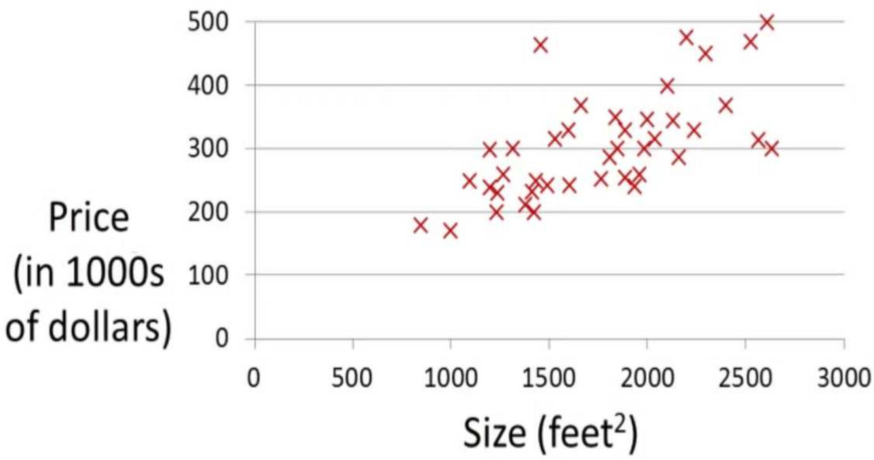


Figura 2.1: Reprezentarea grafică a datelor de vânzare a unor proprietăţi imobiliare. Pe abscisă este măsurată suprafaţa, pe ordonată este preţul [6].

Să presupunem că se doreşte estimarea valorii unei proprietăţi de suprafaţă 1300. Se poate proceda în felul următor: se trasează o dreaptă care să aproximeze "cât mai bine"[1] norul de puncte reprezentat[2]. Eroarea pentru cazul unui punct este influenţată de diferenţa dintre valoarea actuală şi cea prezisă de model – aici modelul este o dreaptă. Se constată apoi care este valoarea de pe dreaptă, corespunzătoare lui 1300 (figura 2.2). Estimarea obţinută este de circa 220000.

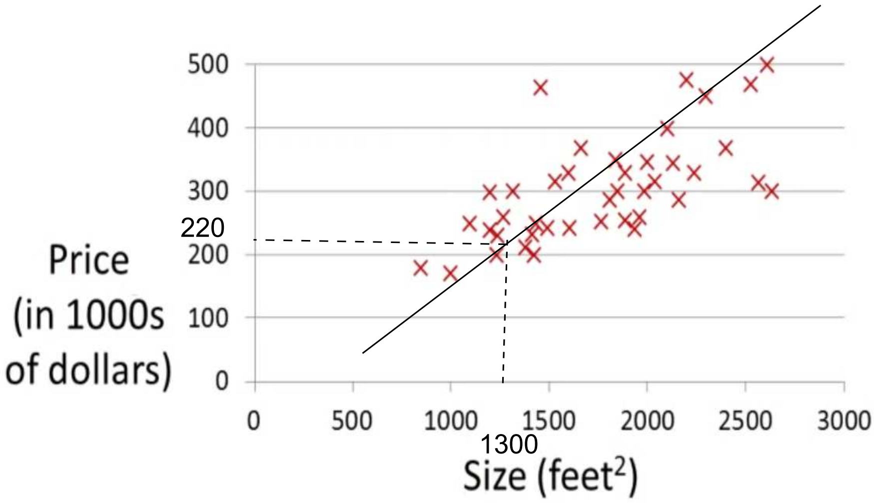


Figura 2.2: Aproximarea preţului pentru o suprafaţă de 1300 feet^2.

Modelul de predicţie figurat mai sus este un model liniar:

$$pret = a \cdot suprafata + b$$

unde a şi b sunt coeficienţi reali ce vor fi determinaţi; a se numeşte pantă (eng: slope) iar b termen liber (eng: intercept). Desigur, se pot folosi forme polinomiale de grad mai mare decât 1, sau modele local liniare, sau reţele neurale etc. Alegerea celui mai bun model pentru un set de date cunoscut este o problemă în sine. Preferarea unui model liniar se motivează prin aceea că în practică se dovedeşte a fi suficient de bun pentru multe probleme, iar modelele mai simple se recomandă a fi încercate printre primele. În plus, un model liniar este uşor de interpretat: creşterea valorii variabilei $suprafata$ cu o unitate duce la creşterea preţului total cu a unităţi monetare; valoarea b este preţul de pornire.

Avem mai sus un exemplu de instruire supervizată: se porneşte de la un set de date cu perechi formate din valoare de intrare (*e.g.* suprafaţa)

[1] O formulă pentru a măsura cât de bună e aproximarea rezultată se va da în secţiunea 2.2.

[2] Pentru cazul cu mai multe date de intrare se obţine o varietate liniară – plan pentru 2 dimensiuni de intrare etc. – dar modul de determinare a ei este similar cu ceea ce se prezintă pentru o singură variabilă.

şi valoare de ieşire asociată (*e.g.* costul suprafeţei). Se cere determinarea unui model care să fie folosit pentru prezicerea (aproximarea) unor valori de ieşire, date fiind valori de intrare furnizate; pentru exemplul considerat, vrem să vedem care e costul estimat al unor suprafeţe.

Formal, într–o problemă de regresie se dau:

- m – numărul de perechi de valori (sau cazuri, sau înregistrări) din setul de instruire; pentru desenul din figura 2.1 este numărul de puncte desenate;
- $\mathbf{x}^{(i)}$ – m vectori de intrare, $1 \leq i \leq m$; un astfel de vector este compus de regulă din n componente numerice, numite trăsături (eng: features): $\mathbf{x}^{(i)} = \left(x_1^{(i)}, x_2^{(i)}, \ldots, x_n^{(i)}\right)^t$: suprafaţa, distanţa la utilităţi etc. Trăsăturile $x_j^{(i)}$ se mai numesc şi variabile predictive sau independente[3];
- $y^{(i)}$, $1 \leq i \leq m$ – variabila de ieşire (sau de predicţie, sau dependentă) aferentă valorii $x^{(i)}$; în cazul exemplificat este un număr real (preţul), dar în general poate fi un vector de valori reale.

Perechea i din setul de antrenare este $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})$, $1 \leq i \leq m$. Întregul set de antrenare se scrie ca:

$$\mathcal{S} = \left\{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}) | 1 \leq i \leq m\right\} \tag{2.1}$$

Setul de antrenare se specifică frecvent sub formă tabelară, precum în tabelul 2.1.

Suprafaţa (picioare pătrate)	Preţul (mii de dolari)
2100	450
1410	243
800	188
...	...

Tabela 2.1: Set de date de instruire

Fluxul de lucru în învăţarea automată[4] este dat în figura 2.3: se porneşte de la un set de instruire, se aplică un algoritm de învăţare şi se produce un model. Din motive istorice acest model se mai numeşte şi ipoteză şi se notează de regulă cu h. Algoritmul de instruire are ca scop determinarea unei forme adecvate a modelului, de exemplu a unor valori potrivite a coeficienţilor funcţiei h.

[3] A nu se confunda cu noţiunea de independenţa liniară din algebră, sau cu independenţa evenimentelor şi a variabilelor aleatoare din teoria probabilităţilor.

[4] În limba engleză: machine learning = învăţare automată.

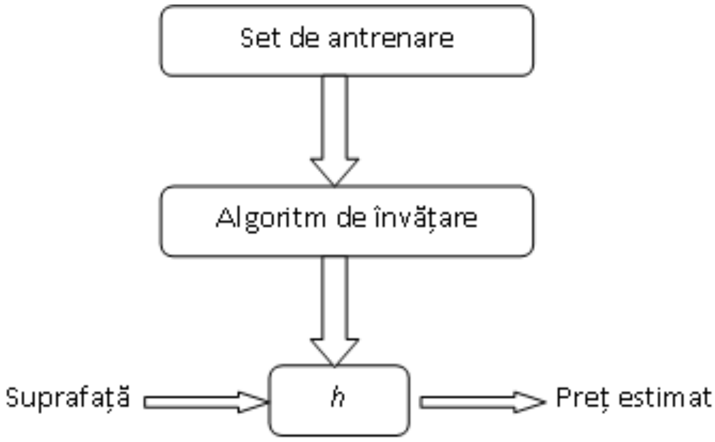


Figura 2.3: Fluxul de lucru într-un proces de instruire automată.

După ce instruirea se termină, modelului rezultat i se furnizează o intrare (în exemplul nostru: suprafaţa) şi modelul va calcula o valoare de ieşire estimată (preţul). În notaţie formală avem ecuaţia 2.2:

$$\hat{\mathbf{y}} = h(\mathbf{x}) \tag{2.2}$$

unde notaţia cu căciulă se foloseşte pentru a denota valori care sunt estimate de un model oarecare.

Una din întrebările esenţiale este: cum se reprezintă ipoteza h? Există mai multe variante ce pot fi utilizate. Mai sus am pornit cu presupunerea că preţul creşte liniar cu suprafaţa vândută, deci:

$$h(x) = h_\theta(x) = \theta_0 + \theta_1 \cdot x \tag{2.3}$$

unde indicele lui h este vectorul coloană $\boldsymbol{\theta} = (\theta_0, \theta_1)^t$.

Acest model (ipoteză) se numeşte regresie liniară cu o variabilă, sau regresie liniară univariată. Se poate ca pe lângă suprafaţă – singura valoare de intrare considerată până acum – să se mai considere şi alte variabile de intrare: distanţa de la proprietate la utilităţi, gradul de poluare a zonei etc.; în acest caz, modelul ar fi unul multivariat (mai multe valori de intrare considerate). Coeficienţii θ_0 şi θ_1 din ecuaţia (2.3) se mai numesc parametri ai modelului de predicţie şi se determină prin pasul de învăţare.

Modelul (2.3) se mai poate scrie astfel:

$$h_\theta(x) = \theta_0 \cdot 1 + \theta_1 \cdot x = \theta_0 \cdot x_0 + \theta_1 \cdot x_1 = \boldsymbol{\theta}^t \cdot \mathbf{x} \tag{2.4}$$

unde: $x_0 = 1$, $x_1 = x$, vectorul $\boldsymbol{\theta}$ este $(\theta_0, \theta_1)^t$, vectorul $\mathbf{x}$ este $(x_0, x_1)^t$.

2.2 Funcţia de cost

Există o infinitate de moduri în care se poate trasa dreapta din figura 2.2; altfel zis, există o infinitate de valori pentru coeficienţii din modelul dat de ecuaţia (2.3).

Se pune problema: cum alegem cât mai bine aceşti coeficienţi? O variantă naturală este determinarea acestora de aşa manieră încât valorile *prezise* de model, $h_\theta(x^{(i)})$, să fie cât mai apropiate de valorile *cunoscute* $y^{(i)}$, pentru tot setul de antrenare $\mathcal{S}$ din (2.1). Pentru toate valorile din setul de instruire, eroarea cumulată se poate măsura cu funcţia de cost

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2 \tag{2.5}$$

Funcţia de eroare J se mai numeşte şi funcţie de cost a modelului[5]. Se pot folosi şi alte funcţii de cost, de exemplu incluzând constrângeri impuse valorilor parametrilor $\boldsymbol{\theta}$. Funcţia de eroare pătratică este o alegere populară pentru problemele de regresie, dar nu singura posibilă. Factorul m de la numitor apare pentru a calcula media erorii (altfel, eroarea ar creşte de fiecare dată când se adaugă în setul de instruire o pereche $(x^{(i)}, y^{(i)})$ pentru care $h_\theta(x^{(i)}) \neq y^{(i)}$, în timp ce media permite compararea erorilor modelului peste seturi de dimensiuni diferite); numitorul 2 se utilizează din motive estetice pentru calculele de mai târziu.

Trebuie să găsim acele valori ale coeficienţilor $\theta_0^{(min)}, \theta_1^{(min)}$ pentru care se atinge minimul funcţiei de eroare:

$$\begin{aligned} \left(\theta_0^{(min)}, \theta_1^{(min)}\right)^t &= \arg \min_{(\theta_0, \theta_1)^t \in \mathbb{R}^2} J(\theta_0, \theta_1) = \\ &= \arg \min_{(\theta_0, \theta_1)^t \in \mathbb{R}^2} \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2 \end{aligned} \tag{2.6}$$

Merită să discutăm comportamentul funcţiei J pentru cazuri particulare. De exemplu, dacă $\theta_0 = 0$, funcţia de eroare $J(0, \theta_1)$ este o funcţie de gradul 2 depinzând de o singură variabilă (θ_1) şi având minimul mai mare sau egal cu zero. Pentru θ_0, θ_1 oarecare forma funcţiei de eroare este dată în figura 2.4 [6].

O altă variantă de reprezentare grafică a funcţiei de eroare este pe baza curbelor de contur: reprezentarea este plană, având pe cele două axe respectiv pe θ_0, θ_1. Pentru o valoare oarecare a funcţiei de eroare se consideră mulţimea tuturor perechilor de parametri θ_0, θ_1 pentru care se obţine aceeasi valoare a erorii. Rezultatul este dat de o mulţime de curbe, precum cele reprezentate în figura 2.5 [6]. Se poate arăta că aceste contururi sunt eliptice.

[5]În limba engleză: loss function, error function, cost function.

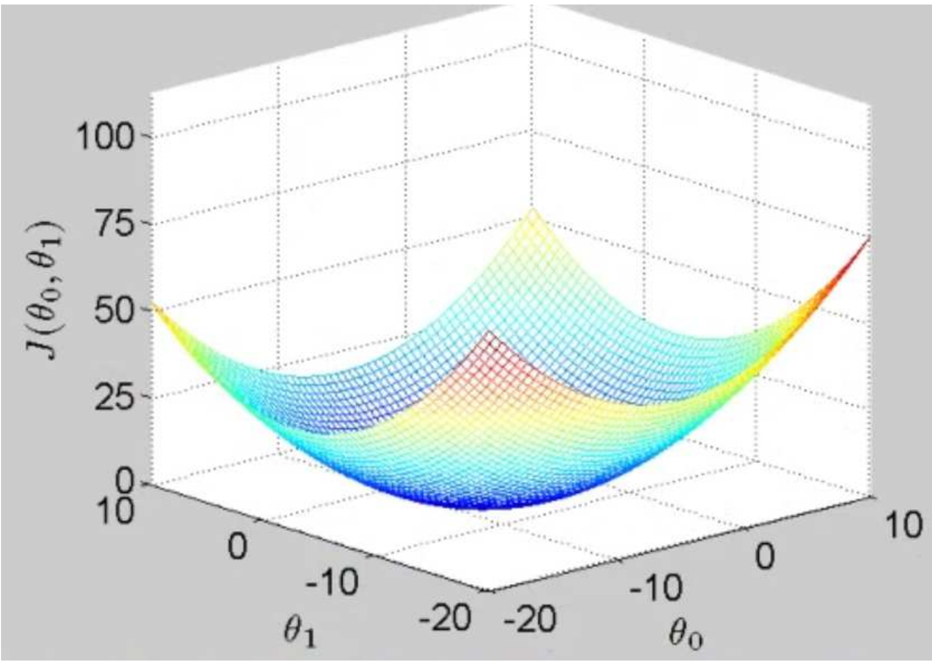


Figura 2.4: Funcţia de eroare pentru model liniar univariat, cu coeficienţi θ_0, θ_1 [6].

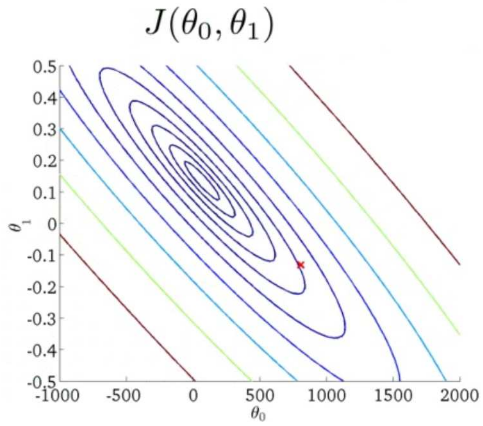


Figura 2.5: Curbe de contur pentru functia de eroare a unui model liniar univariat [6].

2.3 Metoda de căutare după direcţia gradientului

În această secţiune se va prezenta o metodă iterativă (eng: gradient descent) pentru minimizarea funcţiei de eroare J. Ideea e simplă:

- se porneşte cu valori θ_0, θ_1 iniţiale, setate aleator sau chiar 0;
- se modifică în mod iterativ valorile curente ale parametrilor θ_0, θ_1 de aşa manieră încât J să scadă.

Ultimul punct se concretizează astfel: valorile curente ale parametrilor θ_0, θ_1 se modifică conform

$$\theta_0 = \theta_0 - \alpha \cdot \frac{\partial J}{\partial \theta_0}(\theta_0, \theta_1) \tag{2.7}$$

$$\theta_1 = \theta_1 - \alpha \cdot \frac{\partial J}{\partial \theta_1}(\theta_0, \theta_1) \tag{2.8}$$

şi atribuirile se operează în mod simultan pentru θ_0, θ_1.

Această simultaneitate e cerută din cauză că la calculele (2.7–2.9) trebuie să ne asigurăm că aceiaşi θ_0, θ_1 sunt folosiţi pentru evaluarea ambelor derivate parţiale. Simultaneitatea se poate obţine astfel: se calculează expresiile din membrii drepţi ai ecuaţiilor (2.7) şi (2.8) şi se asignează unor variabile temporare $\theta_0^{(temp)}$ şi respectiv $\theta_1^{(temp)}$; doar după ce ambele variabile temporare sunt calculate, valorile lor se atribuie corespunzător: $\theta_0 = \theta_0^{(temp)}$ şi $\theta_1 = \theta_1^{(temp)}$. Alternativ, se poate folosi calcul vectorizat.

Aceleaşi formule se scriu în mod vectorial ca:

$$\begin{pmatrix} \theta_0 \\ \theta_1 \end{pmatrix} = \begin{pmatrix} \theta_0 \\ \theta_1 \end{pmatrix} - \alpha \cdot \begin{pmatrix} \frac{\partial J}{\partial \theta_0}(\theta_0, \theta_1) \\ \frac{\partial J}{\partial \theta_1}(\theta_0, \theta_1) \end{pmatrix} \tag{2.9}$$

Dacă folosim notaţia nabla pentru vectorul de derivate parţiale (vectorul gradient)[6]:

$$\nabla_{\boldsymbol{\theta}} J = \begin{pmatrix} \frac{\partial J}{\partial \theta_0} \\ \frac{\partial J}{\partial \theta_1} \end{pmatrix} \tag{2.10}$$

atunci mai putem scrie formula de modificare a ponderilor ca:

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \alpha \cdot \nabla J_{\boldsymbol{\theta}}(\theta_0, \theta_1) \tag{2.11}$$

În bibliotecile care permit calcul matriceal condiţia de atribuire simultană este respectată automat. O formă matriceală generală pentru gradient este dată în continuare.

[6]Vectorul de derivate parţiale este un vector de funcţii; acestea se vor evalua pentru perechea de valori θ_0, θ_1.

Coeficientul $\alpha > 0$ se numeşte rată de învăţare; poate fi o constantă sau o cantitate care variază de–a lungul iteraţiilor. Alegerea lui α este crucială: dacă valoarea lui e prea mică, atunci algoritmul va face foarte multe iteraţii până se va opri, deci am avea un cost computaţional mare. Dacă e prea mare, procesul poate să rateze minimul sau chiar să diveargă (valoarea lui J să crească mereu sau periodic). Dacă se constată acest al doilea fenomen, valoarea lui α trebuie scăzută. Odată ce o valoare potrivită pentru α este găsită, nu e neapărat nevoie ca aceasta să fie modificată de-a lungul iteraţiilor.

Metoda se poate folosi pentru reducerea valorilor unei funcţii de oricâte variabile. Menţionăm că în general se poate ajunge într-un minim local al funcţiei căreia i se aplică.

Valorile θ_0, θ_1 se iniţializează aleator cu valori mici, sau chiar 0. Algoritmul de căutare după direcţia gradientului are forma:

`repeta{`

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \quad \text{simultan pentru } j = 0, 1 \tag{2.12}$$

`} pana la convergenta`

Criteriul de convergenţă poate fi: de la o iteraţie la alta valoarea lui J nu mai scade semnificativ, sau norma diferenţei între două valori succesive ale vectorului $\boldsymbol{\theta} = (\theta_0, \theta_1)^t$ este sub un prag mic $\varepsilon > 0$ setat, sau se atinge un număr maxim de iteraţii permise.

Putem explicita derivatele parţiale pentru forma funcţiei de eroare considerate şi (2.12) devine:

$$\theta_j := \theta_j - \alpha \cdot \frac{1}{m} \sum_{i=1}^{m} \left[\left(h_\theta(x^{(i)}) - y^{(i)} \right) \cdot x_j^{(i)} \right] \quad \text{simultan pentru } j = 0, 1 \tag{2.13}$$

Deoarece gradientul unei funcţii în extremele funcţiei este vectorul zero (vectorul nul), rezultă că valoarea parametrilor $\boldsymbol{\theta}$ nu se va mai modifica, odată ce s–a atins o valoare de minim – global sau local – a lui J. Această observaţie explică două din condiţiile de convergenţă.

Următoarele sugestii ar trebui să fie luate în considerare:

- valorile trăsăturilor de intrare să fie în scale similare; se recomandă deci a se face în prealabil o scalare a datelor la un interval convenabil ales, *e.g.* $[0, 1]$; ca efect se obţine de regulă un număr mult mai mic de iteraţii până la convergenţa algoritmului;

- se vor urmări valorile lui J; dacă ele au nu o tendinţă descrescătoare (funcţia J creşte sau are scăderi urmate de creşteri) atunci se va încerca o valoare mai mică pentru rata de învăţare α; dacă valoarea funcţiei J scade foarte lent se poate mări valoarea lui α.

Valoarea optimă a lui α depinde de setul de date peste care se calculează funcţiei de eroare J. α este un hiperparametru care influenţează succesul şi viteza învăţării.

Cazul în care datele de intrare sunt multivariate: $\mathbf{x}^{(i)} = \left(x_1^{(i)}, x_2^{(i)}, \dots, x_n^{(i)}\right)^t$ se poate trata tot prin metoda de căutare după direcţia gradientului, prin modificări imediate:

1. modelul de predicţie devine

$$h_{\boldsymbol{\theta}}(\mathbf{x}) = \theta_0 + \theta_1 \cdot x_1 + \dots \theta_n \cdot x_n = \boldsymbol{\theta}^t \cdot \mathbf{x} \tag{2.14}$$

unde $\mathbf{x} = (x_0 = 1, x_1, \dots, x_n)^t$, $\boldsymbol{\theta} = (\theta_0, \theta_1, \dots, \theta_n)^t$. Am făcut trecerea de la vectorul $\mathbf{x}^{(i)} = \left(x_1^{(i)}, x_2^{(i)}, \dots, x_n^{(i)}\right)^t$ de n componente la unul de $n+1$ componente, prin adăugarea unei valori $x_0 = 1$

2. funcţia de eroare J devine:

$$J(\boldsymbol{\theta}) = \frac{1}{2m} \sum_{i=1}^{m} (h_{\boldsymbol{\theta}}(\mathbf{x}_i) - y_i)^2 \tag{2.15}$$

o generalizare a ecuaţiei (2.5), de la două argumente θ_0, θ_1 la vectorul $\boldsymbol{\theta}$ de $n+1$ componente

3. ecuaţia (2.12) din algoritmul de căutare devine:

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\boldsymbol{\theta}) = \theta_j - \alpha \cdot \frac{1}{m} \sum_{i=1}^{m} \left[\left(h_\theta(x^{(i)}) - y^{(i)} \right) \cdot x_j^{(i)} \right] \tag{2.16}$$

simultan pentru $j = 0, 1, \dots, n$.

Rescriem în cele ce urmează funcţia de cost (2.15) şi formula de modificare a ponderilor din ecuaţia (2.16) folosind calcul matriceal. Acest lucru favorizeaza implementare eficientă în medii precum Matlab sau NumPy.

Pentru început, notăm cu $\mathbf{X}$ matricea datelor de intrare:

$$\mathbf{X} = \begin{pmatrix} x_1^{(1)} & x_2^{(1)} & \dots & x_n^{(1)} \\ x_1^{(2)} & x_2^{(2)} & \dots & x_n^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{(m)} & x_2^{(m)} & \dots & x_n^{(m)} \end{pmatrix} \tag{2.17}$$

unde linia $\mathbf{x}^{(i)}$ conţine valorile predictive asociate celui de al i–lea caz din setul de instruire, iar vectorul coloană de indice $1 \leq j \leq n$ corespunde unei trăsături predictive. Valorile de ieşire corespunzătoare sunt de asemenea stocate matriceal, folosind un vector coloană:

$$\mathbf{y} = \begin{pmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{pmatrix} \in \mathbb{R}^m \tag{2.18}$$

Putem extinde matricea de date $\mathbf{X}$ din ecuaţia (2.17) cu o primă coloană plină cu 1; pentru simplitatea notaţiilor, vom folosi şi în continuare litera $\mathbf{X}$ pentru această matrice, numită matrice de design:

$$\mathbf{X} = \begin{pmatrix} 1 & x_1^{(1)} & x_2^{(1)} & \dots & x_n^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} & \dots & x_n^{(2)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_1^{(m)} & x_2^{(m)} & \dots & x_n^{(m)} \end{pmatrix} \tag{2.19}$$

Vom folosi următoarele relaţii cunoscute din algebra liniară:

- transpusa unei sume de matrice este suma transpuselor:

$$(A+B)^t = A^t + B^t \tag{2.20}$$

- transpusa unui produs de matrice este produsul matricelor în ordine inversă:

$$(A_1 \cdot A_2 \dots A_p)^t = A_p^t \cdot A_{p-1}^t \dots A_1^t \tag{2.21}$$

- dacă a este un număr real, atunci el poate fi interpretat ca o matrice cu o linie şi o coloană şi din acest motiv

$$a^t = a \tag{2.22}$$

- conform lucrării [7] secţiunea 2.4.1, ecuaţia (69):

$$\nabla_{\boldsymbol{\theta}} \left(\boldsymbol{\theta}^t \mathbf{A}\right) = \mathbf{A} \tag{2.23}$$

- conform aceleiaşi lucrări secţiunea 2.4.2, ecuaţia (81):

$$\nabla_{\boldsymbol{\theta}} \left(\theta^t \mathbf{A} \theta\right) = (\mathbf{A} + \mathbf{A}^t) \cdot \theta \tag{2.24}$$

  pentru $\mathbf{A}$ (obligatoriu) matrice pătratică.

Funcţia de cost J din ecuaţia (2.15) se rescrie matriceal astfel:

$$\begin{aligned} J(\boldsymbol{\theta}) &= \frac{1}{2m} \sum_{j=1}^{m} \left(h_\theta(\mathbf{x}^{(j)}) - y^{(j)}\right)^2 \\ &= \frac{1}{2m} \sum_{j=1}^{m} \left(\boldsymbol{\theta}^T \mathbf{x}^{(j)} - y^{(j)}\right)^2 \\ &= \frac{1}{2m} (X\boldsymbol{\theta} - \boldsymbol{y})^t (X\boldsymbol{\theta} - \boldsymbol{y}) \end{aligned} \tag{2.25}$$

Pentru calculul matriceal al vectorului de derivate parţiale $\left(\frac{\partial}{\partial\theta_j}J(\theta)\right)_{j=\overline{0,n}}$ dezvoltăm (2.25):

$$\begin{aligned} J(\boldsymbol{\theta}) &= \frac{1}{2m}\left(X\boldsymbol{\theta} - \boldsymbol{y}\right)^t\left(X\boldsymbol{\theta} - \boldsymbol{y}\right) \\ &= \frac{1}{2m}\left\{(\mathbf{X}\boldsymbol{\theta})^t\,\mathbf{X}\boldsymbol{\theta} - (\mathbf{X}\boldsymbol{\theta})^t\,\mathbf{y} - \mathbf{y}^t\left(\mathbf{X}\boldsymbol{\theta}\right) + \mathbf{y}^t\mathbf{y}\right\} \\ &= \frac{1}{2m}\left\{\boldsymbol{\theta}^t\mathbf{X}^t\mathbf{X}\boldsymbol{\theta} - 2\boldsymbol{\theta}^t\mathbf{X}^t\mathbf{y} + \mathbf{y}^t\mathbf{y}\right\} \end{aligned} \tag{2.26}$$

Ţinem cont de faptul că derivata parţială e operator liniar, deci derivata parţială a unei sume de funcţii este suma derivatelor parţiale ale acelor funcţii:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \frac{1}{2m}\left\{\nabla_{\boldsymbol{\theta}}\left(\boldsymbol{\theta}^t\mathbf{X}^t\mathbf{X}\boldsymbol{\theta}\right) - 2\nabla_{\boldsymbol{\theta}}\left(\boldsymbol{\theta}^t\mathbf{X}^t\mathbf{y}\right) + \nabla_{\boldsymbol{\theta}}\left(\mathbf{y}^t\mathbf{y}\right)\right\} \tag{2.27}$$

Considerând relaţiile (2.20–2.24) şi observând că scalarul $\mathbf{y}^t\mathbf{y}$ nu depinde de $\boldsymbol{\theta}$, obţinem vectorul gradient:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \frac{1}{m}\left\{\mathbf{X}^t\mathbf{X}\boldsymbol{\theta} - \mathbf{X}^t\mathbf{y}\right\} = \frac{1}{m}\mathbf{X}^t\left(\mathbf{X}\boldsymbol{\theta} - \mathbf{y}\right) \tag{2.28}$$

Modificările de ponderi θ_j din ecuaţia (2.16) se scriu matriceal pentru vectorul $\boldsymbol{\theta}$ ca:

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \frac{\alpha}{m}\mathbf{X}^t(\mathbf{X}\boldsymbol{\theta} - \mathbf{y}) \tag{2.29}$$

iar rescrierea algoritmului de instruire prin gradient descent este imediată.

2.4 Metoda ecuaţiilor normale

Există o metodă care dă o soluţie pe baza unui calcul algebric.

Valorile căutate pentru $\boldsymbol{\theta}$ sunt cele care produc minimul valorii lui J:

$$\boldsymbol{\theta}^{(min)} = \arg\min_{\boldsymbol{\theta}\in\mathbb{R}^{n+1}} J(\boldsymbol{\theta}) \tag{2.30}$$

Conform teoremei lui Fermat, o condiţie necesară pentru ca $\boldsymbol{\theta}^{(min)}$ să minimizeze pe J este ca vectorul derivatelor parţiale calculat în $\boldsymbol{\theta}^{(min)}$ să fie vectorul nul:

$$\nabla_{\boldsymbol{\theta}} J\left(\boldsymbol{\theta}^{(min)}\right) = \mathbf{0} \tag{2.31}$$

unde $\mathbf{0}$ este vector coloană format din $n+1$ elemente zero.

Deoarece funcţia de eroare J e şi convexă, condiţia necesară de extrem dată de (2.31) este şi suficientă şi deci se ajunge în unicul minim al lui J. Înlocuind formula gradientului $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ din ecuaţia (2.28) în (2.31) obţinem:

$$\mathbf{X}^t\mathbf{X}\boldsymbol{\theta}^{(min)} = \mathbf{X}^t\mathbf{y} \tag{2.32}$$

ce defineşte un sistem de ecuaţii numite "ecuaţiile normale". Mai departe, dacă matricea $\mathbf{X}^t\mathbf{X}$ este nesingulară, vectorul de parametri $\boldsymbol{\theta}^{(min)}$ se determină ca

$$\boldsymbol{\theta}^{(min)} = \left(\mathbf{X}^t\mathbf{X}\right)^{-1}\mathbf{X}^t \cdot \mathbf{y} \tag{2.33}$$

Precizări:

1. Expresia $\left(\mathbf{X}^t\mathbf{X}\right)^{-1}\mathbf{X}^t$ se mai numeşte şi pseudo–inversa Moore–Penrose şi se notează cu $\mathbf{X}^+$; pentru o matrice inversabilă inversa şi pseudo–inversa ei coincid; pentru calculul pseudoinversei unei matrice A se poate folosi în Octave şi Matlab funcţia `pinv`, iar în Python funcţia `numpy.linalg.pinv`;

2. Când se foloseşte metoda ecuaţiilor normale, nu este necesar să se facă scalarea trăsăturilor de intrare, precum se recomandă la metoda iterativă.

Una din problemele care trebuie discutate este: cum se procedează când matricea $\mathbf{X}^t\mathbf{X}$ este singulară? Acest lucru se datorează de regulă uneia din situaţiile de mai jos:

- există trăsături de intrare redundante, de exemplu două coloane ale lui $\mathbf{X}$ sunt liniar dependente; în acest caz avem în mod clar o redundanţă informaţională şi putem elimina oricare din aceste două coloane; mai general, una din coloane poate fi combinaţie liniară a altor coloane şi dacă se ştie care e, se poate elimina;

- se folosesc prea multe trăsături faţă de numărul de cazuri din setul de instruire ($m \leq n$); în acest caz se poate renunţa la câteva trăsături, adică se elimină coloane din $\mathbf{X}$, sau se foloseşte regularizarea – a se vedea secţiunea 2.5.

Ordinea de mai sus este cea sugerată pentru acţionare: se elimină din coloanele redundante, apoi dacă încă e nevoie, se foloseşte regularizarea.

Dat fiind faptul că avem două metode de determinare a lui $\boldsymbol{\theta}^{(min)}$, se pune problema pe care din ele să o preferăm. Iată câteva comparaţii:

1. În timp ce pentru metoda gradient descent trebuie ca rata de învăţare să fie aleasă cu grijă, pentru varianta algebrică aşa ceva nu e necesar, neavând de fapt rată de învăţare;

2. În timp ce pentru metoda de calcul bazată pe gradient descent sunt necesare mai multe iteraţii, metoda algebrică necesită un singur pas;

3. Metoda bazată pe gradient descent funcţionează bine chiar şi pentru valori mari ale lui m şi n; pentru valori m sau n mari, calculul pseudoinversei poate fi prohibitiv din punct de vedere al memoriei şi timpului de calcul necesar.

2.5 Overfitting, underfitting, regularizare

2.5.1 Overfitting, underfitting

Pentru problema estimării preţului unei proprietăţi, să presupunem că există 5 perechi de valori în setul de instruire, o pereche fiind constituită din variabila predictivă $suprafata$ şi variabila de ieşire $pret$. Să considerăm 3 modele de predicţie:

1. polinom de gradul întâi: preţul estimat este de forma:
$$pret = \theta_0 + \theta_1 x \tag{2.34}$$
2. polinom de gradul al doilea:
$$pret = \theta_0 + \theta_1 x + \theta_2 x^2 \tag{2.35}$$
3. polinom de gradul 4:
$$pret = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4 \tag{2.36}$$

unde pentru fiecare caz x este suprafaţa; spunem că am introdus noi trăsături pe baza celor existente, corespunzătoare mai sus cantităţilor x^2, x^3, x^4. Primul model este cel liniar discutat până acum, iar celelalte două sunt modele polonomiale (de grad 2, respectiv 4). Ca şi mai înainte, se pune problema determinării coeficienţilor $\theta_0, \theta_1, \ldots$.

Graficele celor trei forme polinomiale sunt date în figura 2.6 [6]. Putem considera cantităţile x, x^2, x^3, x^4 ca fiind variabile de intrare pe baza cărora se realizează predicţia; faptul că ele provin de la acelaşi x (suprafaţa) este o chestiune secundară.

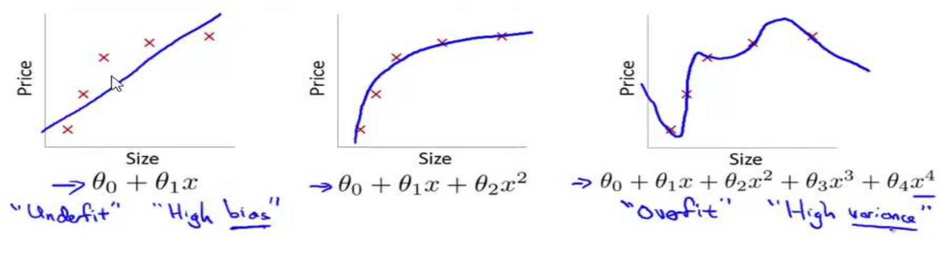


Figura 2.6: Trei polinoame pentru aproximarea preţului pornind de la suprafaţă, notată x [6].

Intuitiv, polinomul de gradul întâi nu reuşeste să facă o aproximare prea bună a evoluţiei preţului faţă de suprafaţă. Spunem că modelul dat de primul polinom suferă de "underfitting"[7], puterea lui de reprezentare fiind

[7]Aproximativ, în limba română: incapacitate de reprezentare

prea slabă pentru problema în cauză. Se mai spune despre un asemenea model că are "high bias"[8], deoarece face o presupunere mult prea simplistă pentru problema tratată.

Pentru polinomul de grad 4, dacă nu se găsesc două preţuri pe aceeaşi verticală (adică nu avem două suprafeţe egale vândute cu preţuri diferite), se pot determina coeficienţii $\theta_0, \ldots, \theta_4$ astfel încât curba să treacă prin toate cele 5 puncte (interpolare polinomială). Remarcăm însă forma nemonotonă, cu variaţii mari a predicţiei, fiind cazuri în care valoarea estimată scade în raport cu suprafaţa. Intuitiv, modelul suferă de "overfitting"[9], fiind prea fidel construit pe datele din setul de instruire; dacă aceste date conţin zgomot, atunci modelul va învăţa perfect zgomotul din setul de date. Deşi reprezentarea pe datele de instruire este perfectă, polinomul de gradul 4 dând chiar preţurile cunoscute, în rest nu face o predicţie prea credibilă (remarcăm scăderea preţului intre al treilea şi al patrulea punct din grafic). Se mai spune că modelul are varianţă (variabililtate) mare[10], datorită faptului că e prea complex pentru problema tratată.

Polinomul de gradul 2 prezintă cea mai credibilă (în sens intuitiv) formă, chiar dacă nu reprezintă exact cele 5 perechi de valori din setul de instruire. Este un compromis între capacitatea de a reproduce setul de instruire şi capacitatea de generalizare, aceasta din urmă fiind abilitatea de a prezice valori de ieşire pentru cazuri care nu fac parte din setul de date de instruire.

Pe scurt, un model care suferă de "underfitting" este incapabil de a reprezenta setul de antrenare, cât şi de a face estimări pentru alte valori. Un model care suferă de "overfitting" poate reprezenta foarte precis datele din setul de instruire, dar nu reuşeşte să facă estimări prea bune în afara lui; în acest ultim caz spunem că nu generalizează bine, generalizarea fiind capacitatea unui model de a estima cât mai aporape de adevăr în afara cazurilor cu care a fost instruit.

Exemplificarea s–a făcut plecând de la o variabilă predictivă x reprezentând suprafaţa, care produce alte valori de predicţie: x^2, x^3, x^4. Mai general, putem să presupunem că avem trăsături de intrare definite în domeniul problemei; în cazul nostru, poate fi distanţa dintre suprafaţa respectivă şi utilităţi, gradul de poluare al zonei etc. Trebuie însă să fim capabili să detectăm cazurile de overfitting şi underfitting şi să le tratăm.

O modalitate de evitare a overfitting–ului este reducerea numărului de trăsături: pentru problema noastră se evită folosirea variabilelor x^3, x^4. O altă variantă este alegerea judicioasă a modelului de predicţie. Cea de a treia opţiune este regularizarea: se păstrează variabilele predictive, dar se impun constrângeri asupra parametrilor modelului — în cazul nostru asupra coeficienţilor θ_i.

[8] Inclinare prea pronunţată spre un model nepotrivit.

[9] Supraspecializare

[10] Engl: high variance.

2.5.2 Regularizare

Să considerăm că predicţia se formează pe baza funcţiei polinomiale

$$h_\theta(\mathbf{x}) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4 \tag{2.37}$$

Intuitiv, vrem ca în funcţia de eroare să includem o constrângere asupra coeficienţilor θ_3, θ_4; ei se înmulţesc cu cantităţile x^3, respectiv x^4 care determină o variaţie rapidă a funcţiei h; altfel zis, o modificare mică a cantităţii x duce la o modificări majore ale lui $h_\theta(x)$. Constrângerea pe care o impunem este deci ca valorile absolute ale lui θ_3 şi θ_4 să fie cât mai apropiate de zero.

Pentru aceasta, vom include în expresia funcţiei de eroare $J(\cdot)$ şi pătratele lui θ_3 şi θ_4:

$$J(\boldsymbol{\theta}) = \left\{ \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2 \right\} + 100 \cdot \theta_3^2 + 100 \cdot \theta_4^2 \tag{2.38}$$

Minimizarea lui J din ec. (2.38) va urmări simultan micşorarea diferenţei dintre valorile estimate de model şi cele reale, dar şi reducerea valorilor absolute ale lui θ_3, θ_4. Exemplul de mai sus este gândit pentru a aduce funcţia polinomială de grad patru la una mai apropiată de gradul al doilea, pentru care agreăm ideea că generalizează mai bine.

În general, nu ştim care dintre coeficienţii care se înmulţesc cu puteri ale lui x ar trebui să aibă valori absolute mici. Intuitiv, ne dăm seama că valoarea lui θ_0 nu ar fi necesar a fi supusă unei constrângeri (nu se înmulţeste cu nicio variabilă predictivă); vom impune deci constrângeri doar asupra lui $\theta_1, \theta_2, \ldots$ – să aibă valori absolute mici. Scopul final este de a evita overfitting–ul. Principiul se aplică şi dacă se pleacă de la variabile de intrare independente, nu neapărat $suprafata, suprafata^2, \ldots$. Ca atare, ec. (2.38) se rescrie mai general ca:

$$J(\boldsymbol{\theta}) = \frac{1}{2m} \left[\sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2 \right] + \lambda \sum_{j=1}^{n} \theta_j^2 \tag{2.39}$$

unde $\lambda > 0$. Cu cât λ e mai mare, cu atât constrângerea impusă coeficienţilor $\theta_1, \ldots, \theta_n$ e mai pronunţată. La extrem, dacă $\lambda \to \infty$ atunci se ajunge la $h_\theta(\mathbf{x}) = \theta_0$, ceea ce aproape sigur înseamnă underfitting (model de aproximare prea simplist): funcţia de estimare returnează mereu θ_0, indiferent de intrare.

Algoritmul de căutare după direcţia gradientului devine (considerăm că avem coeficienţii $\theta_0, \ldots, \theta_n$ iniţializaţi aleator sau chiar cu vectorul nul):

`repeta{`

$$\theta_0 := \theta_0 - \alpha \left[\frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) \cdot x_0^{(i)} \right]$$

$$\theta_j := \theta_j - \alpha \left[\frac{1}{m} \sum_{i=1}^{m} \left(h_{\boldsymbol{\theta}}(x^{(i)}) - y^{(i)} \right) \cdot x_i^{(i)} + 2 \cdot \lambda \cdot \theta_j \right],\ j = 1, \ldots, n$$

```
} pana la convergenta
```

unde atribuirile se efectuează în mod simultan.

Pentru metoda algebrică se poate arăta că regularizarea produce următoarea valoare pentru $\boldsymbol{\theta}$:

$$\boldsymbol{\theta} = \left(\mathbf{X}^t \mathbf{X} + 2 \cdot \lambda \cdot \underbrace{\begin{pmatrix} 0 & 0 & 0 & \ldots & 0 \\ 0 & 1 & 0 & \ldots & 0 \\ 0 & 0 & 1 & \ldots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \ldots & 1 \end{pmatrix}}_{(n+1)\times(n+1)} \right)^{-1} \cdot \mathbf{X} y \tag{2.40}$$

unde matricea care se înmulţeşte cu λ se obţine din matricea unitate de ordinul $n+1$, modificând primul element în 0 (termenul θ_0 nu se regularizează).

Capitolul 3

Regresia logistică

3.1 Încadrare, motivaţie

Regresia logistică este folosită pentru estimare de probabilitate condiţionată şi clasificare. Iniţial dezvoltată pentru lucrul cu două clase, a fost extinsă pentru a discrimina între oricâte clase — regresia logistică multinomială.

Ca mod de instruire se foloseşte învăţarea supervizată. Intrările sunt vectori numerici, iar clasele sunt fie două (pentru regresia logistică binară), fie mai multe (pentru regresia logistică multinomială).

Exemple de probleme de clasificare cu două clase, tratate de regresia logistică, sunt:

- clasificarea unui email ca fiind de tip spam sau nonspam, dându–se conţinutul lui, subiectul emailului, faptul că expeditorul face sau nu parte din lista de contacte etc.
- clasificarea unei tumori ca fiind benignă sau malignă, date fiind rezultatele unor analize;
- clasificarea unei imagini: conţine sau nu un anumit animal.

Exemple de probleme pentru care există mai mult de două clase sunt:

- clasificarea unui email ca fiind de tip: ştiri, muncă, prieteni, anunţuri, spam etc.;
- clasificarea unui pixel dintr–o imagine ca aparţinând unui măr, pară, banană, cireaşă sau fundal.

Modelul dat de regresia logistică (fie ea binară sau multinomială) construieşte o estimare a probabilităţii condiţionate, dată fiind intrarea curentă (conţinut email, imagine etc.); mai precis, se determină care este probabilitatea ca obiectul descris de vectorul de intrare să fie dintr-o clasă anume:

$$P(clasa_i|intrare) \tag{3.1}$$

Faptul că se estimează probabilităţi, adică valori continue din $[0,1]$ justifică cuvântul "regresie" din denumirile modelului. Clasificarea se face prin determinarea acelei $clase_i$ pentru care probabilitatea din ecuaţia (3.1) este maximă.

3.2 Regresia logistică binară

3.2.1 Setul de instruire

În cazul regresiei logistice binare se urmăreşte discriminarea între două clase. Clasele sunt convenabil date ca fiind "1" – clasa pozitivă – şi respectiv "0"– clasa negativă. Setul de instruire este de forma:

$$\mathcal{S} = \left\{(\mathbf{x}^{(j)}, y^{(j)}) | 1 \leq j \leq m\right\} \tag{3.2}$$

unde vectorul $\mathbf{x}^{(j)} = \left(x_0^{(j)}, x_1^{(j)}, \ldots, x_n^{(j)}\right)^t \in \mathbb{R}^{n+1}$ conţine valorile trăsăturilor obiectului j, iar $y^{(j)} \in \{0, 1\}$ este clasa de care aparţine obiectul j. Ca şi până acum, simbolul ^{t} reprezintă transpunerea unui vector sau a unei matrice. Vom considera că $x_0^{(j)} = 1$ pentru orice j, pentru a permite un termen liber în discriminatorul implementat de regresia logistică.

3.2.2 Reprezentarea modelului

Pentru regresia logistică modelul de predicţie trebuie să producă o valoare reprezentând probabilitatea condiţionată (3.1). Vom folosi în acest scop o funcţie $h_{\boldsymbol{\theta}}(\cdot)$ cu proprietatea $0 < h_{\boldsymbol{\theta}}(\mathbf{x}) < 1$:

$$P(y = 1 | \mathbf{x}; \boldsymbol{\theta}) = h_{\boldsymbol{\theta}}(\mathbf{x}) = \frac{1}{1 + \exp(-\boldsymbol{\theta}^t \cdot \mathbf{x})} \tag{3.3}$$

unde $\boldsymbol{\theta} = (\theta_0, \theta_1, \ldots, \theta_n)^t \in \mathbb{R}^{n+1}$ este un vector de $n + 1$ coeficienţi – sau ponderi – ce vor fi determinaţi prin procesul de învăţare. Probabilitatea din (3.3) este o probabilitate condiţionată de intrarea curentă – în cazul nostru: vectorul $\mathbf{x}$ – şi parametrizată de $\boldsymbol{\theta}$. $P(y = 1|\mathbf{x}; \boldsymbol{\theta})$ este gradul de încredere că obiectul descris de vectorul $\mathbf{x}$ face parte din clasa 1, dar modelul este totodată influenţat de parametrul (ponderile din) $\boldsymbol{\theta}$.

Funcţia care stă la baza definirii modelului $h_{\boldsymbol{\theta}}$ este:

$$\sigma : \mathbb{R} \to (0, 1),\ \sigma(z) = \frac{1}{1 + \exp(-z)} \tag{3.4}$$

şi numită sigmoida logistică; este reprezentată grafic în figura 3.1. După cum se remarcă, codomeniul funcţiei logistice este $(0, 1)$, deci compatibil cu valorile admisibile pentru funcţie de probabilitate; extremele 0 şi 1 care sunt permise pentru probabilităţi în general nu se ating însă de către sigmoida logistică. Avem că funcţia σ este derivabilă, strict crescătoare şi

$\lim_{z\to-\infty}\sigma(z) = 0$, $\lim_{z\to\infty}\sigma(z) = 1$. Denumirea de “sigmoidă” este dată de alura graficului, amintind de litera S.

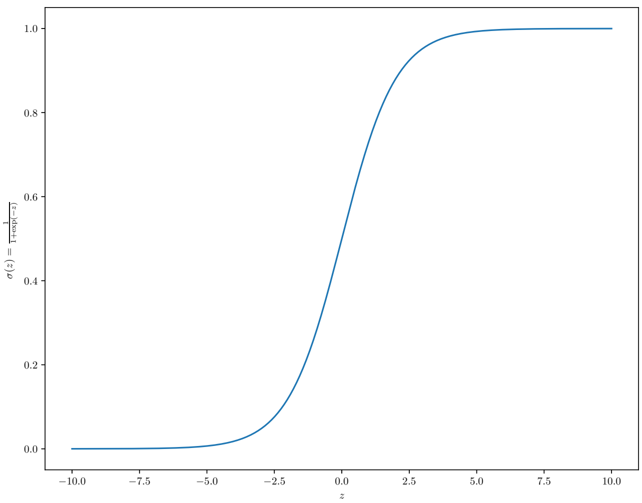


Figura 3.1: Graficul sigmoidei logistice definită în ecuaţia 3.4

Probabilitatea evenimentului ca obiectul $\mathbf{x}$ să fie de clasă 0, negativă, este $P(y = 0|\mathbf{x};\boldsymbol{\theta})$ și e determinată ca find complementul faţă de 1 al evenimentului de a fi de clasă pozitivă[1]:

$$P(y = 0|\mathbf{x};\boldsymbol{\theta}) = 1 - P(y = 1|\mathbf{x};\boldsymbol{\theta}) = \frac{\exp(-\boldsymbol{\theta}^t \cdot \mathbf{x})}{1 + \exp(-\boldsymbol{\theta}^t \cdot \mathbf{x})} \tag{3.5}$$

Dorim să determinăm ponderile (coeficienţii) din vectorul $\boldsymbol{\theta}$ astfel încât pentru acei vectori $\mathbf{x}^{(i)}$ pentru care eticheta asociată este 1 să avem $P(y = 1|\mathbf{x};\boldsymbol{\theta})$ cât mai aproape de 1; pentru $\mathbf{x}^{(i)}$ cu eticheta asociată 0 să avem valoarea $P(y = 0|\mathbf{x};\boldsymbol{\theta})$ cât mai apropiată de 1. Ponderile din vectorul $\boldsymbol{\theta}$ vor fi determinate prin învăţare automată.

După învăţare, modelul probabilist dat de (3.3) este mai departe folosit pentru a face clasificare, astfel: dacă pentru un vector de intrare $\mathbf{x}$ avem că:

$$P(y = 1|\mathbf{x};\boldsymbol{\theta}) \geq P(y = 0|\mathbf{x};\boldsymbol{\theta}) \tag{3.6}$$

atunci se estimează că obiectul descris de vectorul $\mathbf{x}$ este din clasa 1 (pozitivă), altfel din clasa 0 (negativă).

[1] E vorba de două evenimente complementare: un obiect e fie de clasă pozitivă, fie negativă. Sumele probabilităţilor acestor două evenimente este 1, conform axiomelor care fundamentează teoria probabilităţilor.

Având în vedere că $P(y = 1|\mathbf{x}; \boldsymbol{\theta}) + P(y = 0|\mathbf{x}; \boldsymbol{\theta}) = 1$, decizia bazată pe inecuaţia (3.6) se reformulează echivalent ca: vom clasifica obiectul descris de vectorul $\mathbf{x}$ ca fiind de clasă pozitivă dacă $P(y = 1|\mathbf{x}; \boldsymbol{\theta}) \geq 0.5$ şi negativă altfel. Se poate însă să se folosească şi alt prag de discriminare pentru clase, de exemplu: obiectul descris de $\mathbf{x}$ este de clasă pozitivă dacă $P(y = 1|\mathbf{x}; \boldsymbol{\theta}) \geq 0.8$ şi de clasă negativă altfel. O astfel de setare de prag se efectuează pentru seturi de date în care clasele pozitive şi cele negative sunt puternic debalansate (mult mai multe exemple de tip pozitiv decât negativ, sau invers), sau pentru cazul în care penalizarea care se plăteşte pentru o clasificare eronată de tip pozitiv (un fals pozitiv) este foarte mare faţă de penalizarea pentru clasificarea eronată de tip negativ (fals negativ) – sau invers. Pentru ultimul caz, un exemplu este: e mai grav dacă un mail legitim este clasificat ca fiind de tip spam şi scos din inbox, faţă de cazul în care un mail spam este clasificat eronat ca fiind legitim: vom impune ca $P(spam|email)$ sa fie comparat cu un prag t mare (de exemplu 0.9, 0.99) pentru a decide că e vorba într–adevăr de spam.

3.2.3 Suprafaţa de decizie a regresiei logistice

În pofida caracterului neliniar al funcţiei ce defineşte modelul – conform ecuaţiei (3.3) – se arată uşor că suprafaţa care desparte regiunea $\mathbf{x}$ de clasă pozitivă şi cea de clasă negativă este o varietate liniară[2], dacă pragul este 0.5.

Inegalitatea (3.6) se scrie echivalent:

$$\begin{aligned}
P(y = 1|\mathbf{x}; \boldsymbol{\theta}) \geq P(y = 0|\mathbf{x}; \boldsymbol{\theta}) \quad &\Longleftrightarrow \quad \frac{1}{1+\exp(-\boldsymbol{\theta}^t \cdot \mathbf{x})} \geq \frac{\exp(-\boldsymbol{\theta}^t \cdot \mathbf{x})}{1+\exp(-\boldsymbol{\theta}^t \cdot \mathbf{x})} \\
&\Longleftrightarrow \quad 1 \geq \exp(-\boldsymbol{\theta}^t \cdot \mathbf{x}) \\
&\overset{\text{(logaritmând)}}{\Longleftrightarrow} \quad 0 \geq -\boldsymbol{\theta}^t \cdot \mathbf{x} \\
&\Longleftrightarrow \quad \boldsymbol{\theta}^t \cdot \mathbf{x} \geq 0
\end{aligned} \tag{3.7}$$

Am obţinut deci că dacă $\mathbf{x}$ are proprietatea că $\boldsymbol{\theta}^t \cdot \mathbf{x} \geq 0$ atunci $\mathbf{x}$ este clasificat ca fiind de clasă 1, altfel este de clasă 0. Separarea dintre cele două clase se face de către varietatea liniară $\boldsymbol{\theta}^t \cdot \mathbf{x} = 0$: dacă $\mathbf{x}$ e în partea pozitivă a varietăţii liniare $\boldsymbol{\theta}^t \cdot \mathbf{x} = 0$ sau chiar pe această varietate, atunci e de clasă 1, altfel e de clasă 0.

Unii autori consideră că dacă avem o intrare $\mathbf{x}$ pentru care $P(y = 1|\mathbf{x}; \boldsymbol{\theta}) = P(y = 0|\mathbf{x}; \boldsymbol{\theta}) = 0.5$, atunci modelul nu ar trebui să facă o clasificare a intrării: este tot atât de probabil să fie de clasă pozitivă pe cât e de probabil să fie de clasă negativă. În inecuaţia (3.7) am considerat – în mod mai degrabă arbitrar – că situaţia cu egalitate să fie tratată ca un caz pozitiv.

[2]Prin abuz se foloseşte şi denumirea “hiperplan”; în timp ce un hiperplan obligatoriu trebuie să treacă prin origine, varietatea liniară este o formă liniară fără această constrângere.

Dacă se permite ca în componenţa vectorului $\mathbf{x}$ să intre şi forme pătratice, cubice etc. ale trăsăturilor originare, atunci suprafaţa de decizie poate fi mai complicată. De exemplu, să considerăm că vectorul de intrare $\mathbf{x}$ este $\mathbf{x} = (x_0 = 1, x_1, x_2, x_1^2, x_2^2, x_1x_2)^t$; rezultă că $\boldsymbol{\theta} \in \mathbb{R}^6$; avem că $\boldsymbol{\theta}^t \cdot \mathbf{x} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2 + \theta_5 x_1 x_2$. Pentru valorile[3] $\theta_0 = -4, \theta_1 = \theta_2 = \theta_5 = 0$, $\theta_3 = \theta_4 = 1$ se obţine ecuaţia suprafeţei de decizie $x_1^2 + x_2^2 = 4$, reprezentând un cerc; în funcţie de poziţia faţă de cerc (înăuntrul sau în afara lui), obiectul de coordonate $(x_1, x_2)^t$ este estimat ca fiind de o clasă sau de cealaltă. Am arătat pe acest caz particular deci că suprafaţa de separare poate fi neliniară, dacă se introduc trăsături suplimentare bazate pe trăsăturile originare.

3.2.4 Funcţia de cost

Funcţia care ne permite să estimăm cât de bun este un vector de ponderi $\boldsymbol{\theta}$ şi care e de asemenea utilizată pentru ajustarea acestor ponderi în procesul de instruire este notată tradiţional cu $J(\cdot)$, $J : \mathbb{R}^{n+1} \to \mathbb{R}_+$, argumentul ei fiind vectorul de ponderi $\boldsymbol{\theta}$. Valoarea se va calcula peste setul de instruire $\mathcal{S}$ din ecuaţia (3.2).

O variantă este dată de utilizarea aceleiaşi funcţii de eroare din capitolul de regresie liniară, ecuaţia (2.25) pagina 25. Se arată însă că pentru problema estimării de probabilitate condiţionată, dată fiind forma funcţiei (modelului) $h_{\boldsymbol{\theta}}$ din ecuaţia (3.3), funcţia de eroare nu mai este convexă şi în acest caz o căutare bazată pe gradient se poate opri într–un minim local — situaţie exemplificată în figura 3.2.

Vom defini J de aşa manieră încât să fie convexă:

$$J(\boldsymbol{\theta}) = \frac{1}{m} \sum_{j=1}^{m} Cost(h_{\boldsymbol{\theta}}(\mathbf{x}^{(j)}), y^{(j)}) \tag{3.8}$$

unde $Cost(h_{\boldsymbol{\theta}}(\mathbf{x}^{(j)}), y^{(j)})$ vrem să îndeplinească următoarele condiţii:

1. (condiţia de apropiere) dacă $h_{\boldsymbol{\theta}}(\mathbf{x})$ şi y sunt valori apropiate, atunci valoarea lui $Cost(h_{\boldsymbol{\theta}}(\mathbf{x}), y)$ trebuie să fie apropiată de 0, şi reciproc;

2. (condiţia de depărtare) dacă $h_{\boldsymbol{\theta}}(\mathbf{x})$ şi y sunt valori îndepărtate, atunci valoarea lui $Cost(h_{\boldsymbol{\theta}}(\mathbf{x}), y)$ trebuie să fie mare, şi reciproc;

Definim convenabil funcţia $Cost(\cdot, \cdot)$ astfel:

$$Cost(h_{\boldsymbol{\theta}}(\mathbf{x}), y) = \begin{cases} -\ln h_{\boldsymbol{\theta}}(\mathbf{x}) & \text{dacă } y = 1 \\ -\ln(1 - h_{\boldsymbol{\theta}}(\mathbf{x})) & \text{dacă } y = 0 \end{cases} \tag{3.9}$$

logaritmii fiind în baza naturală.

[3] În mod normal, valorile lui $\boldsymbol{\theta}$ se determină prin proces de instruire.

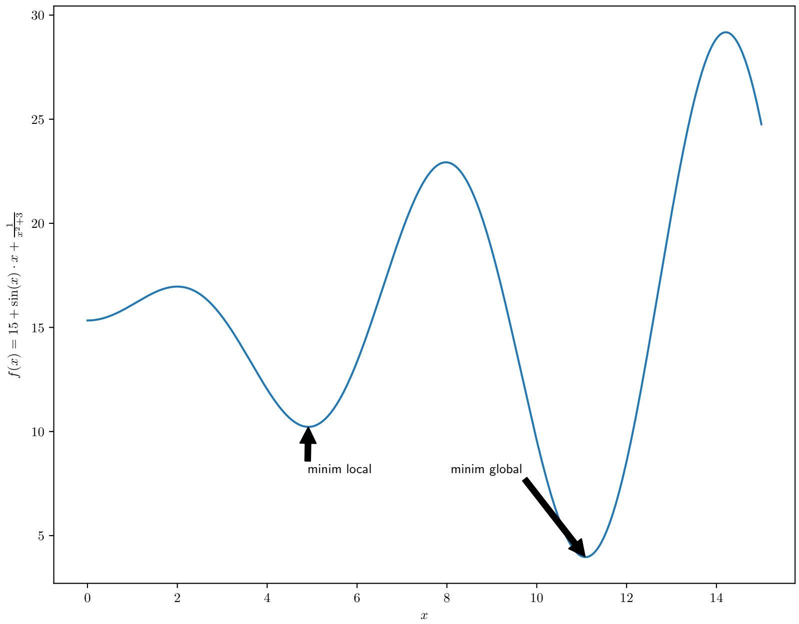


Figura 3.2: Minim global şi minim local pentru funcţia $f : [0, 15] \rightarrow \mathbb{R}$, $f(x) = 15 + \sin(x) \cdot x + \frac{1}{x^2+3}$

Cele două ramuri ale funcţiei *Cost* sunt reprezentate în figura 3.3. Dreptele $x = 0$ şi respectiv $x = 1$ sunt asimptote verticale pentru cele două ramuri ale lui *Cost*.

Rescriem funcţia *Cost* sub forma:

$$Cost(h_{\boldsymbol{\theta}}(\mathbf{x}), y) = -y \cdot \ln h_{\boldsymbol{\theta}}(\mathbf{x}) - (1 - y) \cdot \ln(1 - h_{\boldsymbol{\theta}}(\mathbf{x})) \tag{3.10}$$

Să verificăm că se îndeplinesc cele două condiţii cerute mai sus. Pentru condiţia de apropiere, vrem să verificăm că:

$$Cost(h_{\boldsymbol{\theta}}(\mathbf{x}), y) \approx 0 \Leftrightarrow h_{\boldsymbol{\theta}}(\mathbf{x}) \approx y \tag{3.11}$$

Pentru cazul $y = 1$ avem:

$$Cost(h_{\boldsymbol{\theta}}(\mathbf{x}), y) \approx 0 \Leftrightarrow -\ln h_{\boldsymbol{\theta}}(\mathbf{x}) \approx 0 \Leftrightarrow h_{\boldsymbol{\theta}}(\mathbf{x}) \approx 1 = y \Leftrightarrow h_{\boldsymbol{\theta}}(\mathbf{x}) \approx y$$

Pentru cazul $y = 0$, (3.11) devine:

$$Cost(h_{\boldsymbol{\theta}}(\mathbf{x}), y) \approx 0 \Leftrightarrow -\ln(1 - h_{\boldsymbol{\theta}}(\mathbf{x})) \approx 0 \Leftrightarrow h_{\boldsymbol{\theta}}(\mathbf{x}) \approx 0 = y \Leftrightarrow h_{\boldsymbol{\theta}}(\mathbf{x}) \approx y$$

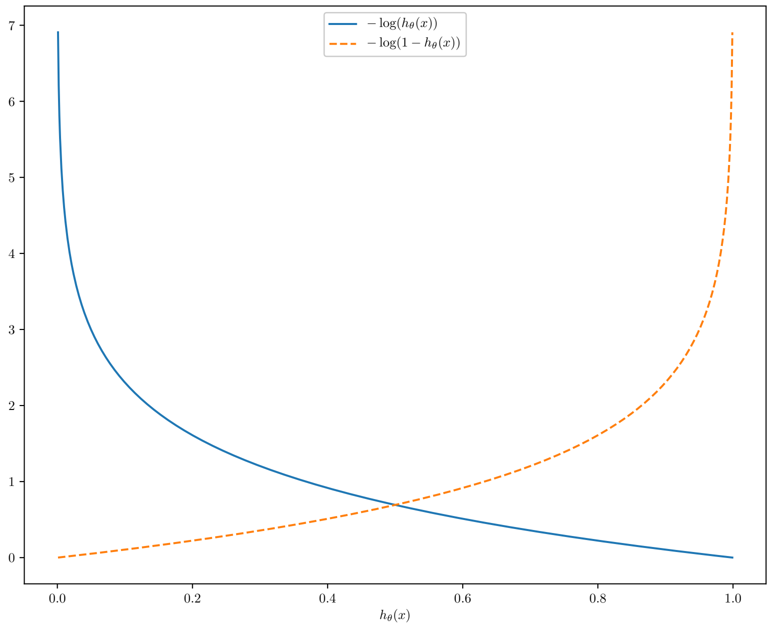


Figura 3.3: Cele două ramuri ale funcţiei $Cost$ din ecuaţia (3.9)

deci prima condiţie, legată de valoarea apropiată de zero a cantităţii $Cost(h_{\boldsymbol{\theta}}(\mathbf{x}), y)$ este îndeplinită de definiţia funcţiei $Cost$ din ecuaţia (3.10).

Pentru condiţia de depărtare, dacă $Cost(h_{\boldsymbol{\theta}}(\mathbf{x}), y) = M \gg 0$, atunci obţinem echivalent $h_{\boldsymbol{\theta}}(\mathbf{x}) = \exp(-M)$ (pentru $y = 1$) respectiv $h_{\boldsymbol{\theta}}(\mathbf{x}) = 1 - \exp(-M)$ (pentru $y = 0$); y este unul din capetele intervalului $[0, 1]$, iar dacă M este o valoare din ce în ce mai mare, atunci $h_{\boldsymbol{\theta}}(\mathbf{x})$ se îndreaptă spre celălalt capăt al intervalului unitate. Rezultă deci că şi această a doua condiţie este îndeplinită de definiţia din formula (3.10).

În plus, deoarece fiecare din cele două ramuri ale funcţiei de cost din (3.9) sunt convexe, rezultă că de fapt funcţia $Cost$ este convexă (ea e una din cele două ramuri, în funcţie de valoarea lui $y^{(j)}$). Mai departe, funcţia J, ca sumă a unui număr finit de funcţii convexe, este ea însăşi convexă. Ca atare, orice punct de minim al lui J este garantat şi minim global; acest lucru este deosebit de util şi justifică forma aparte a funcţiei de cost din (3.9).

Funcţia de eroare J se rescrie astfel:

$$J(\boldsymbol{\theta}) = -\frac{1}{m}\sum_{j=1}^{m}\left[y^{(j)} \cdot \ln h_{\boldsymbol{\theta}}\left(\mathbf{x}^{(j)}\right) + \left(1 - y^{(j)}\right) \cdot \ln\left(1 - h_{\boldsymbol{\theta}}\left(\mathbf{x}^{(j)}\right)\right)\right] \quad (3.12)$$

şi acestă formă se numeşte binary cross–entropy.

3.2.5 Algoritmul de instruire

Setul de antrenare $\mathcal{S}$ este utilizat pentru a deduce valori adecvate ale lui $\boldsymbol{\theta}$, astfel încât predicţiile $h_{\boldsymbol{\theta}}(\mathbf{x}^{(j)})$ date de model pentru valorile de intrare $\mathbf{x}^{(j)}$ să fie cât mai apropiate de valorile actuale ale etichetelor corespunzătoare $y^{(j)}$. Datorită proprietăţilor funcţiei $Cost$ din ecuaţia (3.9) şi a faptului că J este valoarea medie a funcţiei $Cost$ peste setul de instruire, deducem că minimizând valoarea lui J, obţinem (în medie) valori $h_{\boldsymbol{\theta}}(x^{(j)})$ apropiate de $y^{(j)}$.

Pentru determinarea lui $\boldsymbol{\theta}^{(min)}$ care minimizează funcţia J:

$$\boldsymbol{\theta}^{(min)} = \arg \min_{\boldsymbol{\theta} \in \mathbb{R}^{n+1}} J(\boldsymbol{\theta}) \tag{3.13}$$

se foloseşte algoritmul de căutare după direcţia gradientului: se porneşte cu valori aleator setate componentelor vectorului $\boldsymbol{\theta}$ – sau chiar cu vectorul nul – şi se modifică în mod iterativ, scăzând la fiecare pas valoarea gradientului înmulţită cu un coeficient pozitiv mic α, numit rată de învăţare.

Algoritmul de instruire are forma:

1. Setează componentele lui $\boldsymbol{\theta}$ la valori iniţiale aleatoare sau zero;

2. Iteraţii:

   `repeta{`

   $$\theta_i := \theta_i - \alpha \cdot \frac{\partial J}{\partial \theta_i}(\boldsymbol{\theta}) \quad \text{simultan pentru } i = 0, 1, \dots, n$$

   `} pana la convergenta`

Condiţia de convergenţă este la fel ca la regresia liniară.

Explicitând derivatele parţiale ale lui J obţinem:

`repeta{`

$$\theta_i := \theta_i - \alpha \frac{1}{m} \sum_{j=1}^{m} \left(h_\theta(x^{(j)}) - y^{(j)} \right) \cdot x_i^{(j)} \quad \text{simultan pentru } i = 0, \dots, n$$

`} pana la convergenta`

Se observă că modificarea ponderilor $\theta_0, \dots, \theta_n$ are aceeaşi formă ca la regresia liniară. Singura diferenţă este forma funcţiei $h_{\boldsymbol{\theta}}$.

Nu există o formă analitică de determinare a coeficienţilor, cum aveam la regresia liniară.

3.2.6 Regularizare

Dacă se permite ca valorile parametrilor[4] $\theta_1, \ldots, \theta_n$ să fie lăsate neconstrânse, atunci valorile lor absolute pot creşte şi influenţa negativ performanţa de generalizare a modelului: pentru variaţii mici ale datelor de intrare vom avea variaţii mari ale valorilor funcţiei model; mai mult, dacă luăm în considerare trăsături de intrare de forma $x_i \cdot x_j$, $x_i \cdot x_j \cdot x_k$ etc. modelul poate ajunge să aproximeze foarte bine perechile din setul de instruire, dar fără a avea o performanţă bună pe datele din set de testare[5]. Ca atare, se preferă impunerea unor constrângeri parametrilor $\theta_1, \ldots, \theta_n$ astfel încât aceştia să fie cât mai mici în valoare absolută.

Pentru regularizare se modifică forma funcţiei de eroare astfel:

$$J(\boldsymbol{\theta}) = -\frac{1}{m}\sum_{j=1}^{m}\left[y^{(j)} \cdot \ln h_{\boldsymbol{\theta}}(\mathbf{x}^{(j)}) + (1-y^{(j)}) \cdot \ln(1-h_{\boldsymbol{\theta}}(\mathbf{x}^{(j)}))\right] + \frac{\lambda}{2m}\sum_{i=1}^{n}\theta_i^2 \tag{3.14}$$

Modificarea adusă algoritmului de instruire este simplă: mai trebuie inclusă şi derivata parţială a lui θ_i^2 în raport cu θ_i:

```
repeta{
```

$$\theta_0 := \theta_0 - \alpha\frac{1}{m}\sum_{j=1}^{m}(h_\theta(x^{(j)}) - y^{(j)}) \cdot x_0^{(j)}$$

$$\theta_i := \theta_i - \alpha\left[\frac{1}{m}\sum_{j=1}^{m}\left(h_\theta(x^{(j)}) - y^{(j)}\right) \cdot x_i^{(j)} + \frac{\lambda}{m}\theta_i\right]$$

```
} pana la convergenta
```

unde atribuirile se fac simultan pentru inidicii 0, 1, ..., n ai componentelor vectorului $\boldsymbol{\theta}$.

3.3 Regresia logistică multinomială

Pentru cazul în care se cere discriminarea pentru K clase, $K > 2$, regresia logistică se extinde să construiască K funcţii (modele) simultan[6]:

$$h_{\mathbf{\Theta}}(\mathbf{x}) = \begin{pmatrix} P(y=1|\mathbf{x};\mathbf{\Theta}) \\ P(y=2|\mathbf{x};\mathbf{\Theta}) \\ \vdots \\ P(y=K|\mathbf{x};\mathbf{\Theta}) \end{pmatrix} \tag{3.15}$$

[4]Se remarcă lipsa termenului liber θ_0; el nu este regularizat.

[5]Alfel zis: creştem numărul de trăsături din setul de antrenare, dar numărul de exemplare din acest set rămâne acelaşi: m.

[6]Pentru două clase e suficientă construirea unei singure funcţii, sigmoida logistică. Pentru $K > 2$ clase ar fi suficiente construirea a $K-1$ funcţii, dar pentru comoditate se preferă construirea a K funcţii, oricare din ele fiind acceptată ca redundantă.

3.3.1 Setul de instruire

Setul de instruire suferă modificări doar pentru valorile lui $y^{(i)}$, care acum pot fi din mulţimea $\{1, \ldots, K\}$. Formal, setul de instruire se dă ca:

$$\mathcal{S} = \left\{(\mathbf{x}^{(j)}, y^{(j)})|1 \leq j \leq m\right\} \tag{3.16}$$

unde, ca şi la regresia logistică binară, $\mathbf{x}^{(j)} \in \mathbb{R}^{n+1}$, iar $y^{(j)} \in \{1, \ldots, K\}$.

3.3.2 Funcţia softmax

În cazul regresiei logistice cu două clase s–a folosit sigmoidă logistică, producându–se valori de ieşire ce pot fi folosite direct ca probabilităţi condiţionate. Pentru mai mult de două clase, estimarea de probabilitate condiţionată se face cu funcţia *softmax*, care transformă un vector oarecare de K numere într-un vector de K valori din intervalul $(0, 1)$ şi care însumate dau 1.

Pornind de la vectorul $\mathbf{z} = (z_1, \ldots, z_K)^t \in \mathbb{R}^K$, el e transformat de funcţia *softmax* astfel:

$$softmax(\mathbf{z}) = (softmax(\mathbf{z}; 1), \ldots, softmax(\mathbf{z}; K))^t \tag{3.17}$$

unde $softmax(\mathbf{z}; l)$ este:

$$softmax(\mathbf{z}; l) = \frac{\exp(z_l)}{\sum\limits_{k=1}^{K} \exp(z_k)} \tag{3.18}$$

pentru $1 \leq l \leq K$. Se verifică uşor că $softmax(\mathbf{z}; k) \in (0, 1)$ pentru orice k, $1 \leq k \leq K$ şi $\sum_{k=1}^{K} softmax(\mathbf{z}; k) = 1$. Vom folosim funcţia *softmax* pentru producerea de estimări de probabilităţi condiţionate de intrarea curentă $\mathbf{x}$.

3.3.3 Reprezentarea modelului

Ponderile instruibile care definesc modelul de clasificare se grupează într–o matrice $\mathbf{\Theta} \in \mathbb{R}^{(n+1)\times K}$, construită ca:

$$\mathbf{\Theta} = \begin{bmatrix} | & | & | & | \\ \boldsymbol{\theta}_1 & \boldsymbol{\theta}_2 & \cdots & \boldsymbol{\theta}_K \\ | & | & | & | \end{bmatrix} \tag{3.19}$$

unde vectorul $\boldsymbol{\theta}_k \in \mathbb{R}^{n+1}$ conţine parametrii (ponderile instruibile) pentru clasa k, $1 \leq k \leq K$.

Modelul care conţine toate cele K modele, câte unul pentru fiecare clasă, are forma:

$$h_{\mathbf{\Theta}}(\mathbf{x}) = \begin{pmatrix} P(y=1|\mathbf{x};\mathbf{\Theta}) \\ P(y=2|\mathbf{x};\mathbf{\Theta}) \\ \vdots \\ P(y=K|\mathbf{x};\mathbf{\Theta}) \end{pmatrix} = softmax \begin{pmatrix} \boldsymbol{\theta}_1^t \cdot \mathbf{x} \\ \boldsymbol{\theta}_2^t \cdot \mathbf{x} \\ \vdots \\ \boldsymbol{\theta}_K^t \cdot \mathbf{x} \end{pmatrix} \tag{3.20}$$

Probabilitatea ca un vector $\mathbf{x}$ să fie de clasă k $(1 \leq k \leq K)$ este:

$$P(y=k|\mathbf{x};\mathbf{\Theta}) = \frac{\exp\left(\boldsymbol{\theta}_k^t \cdot \mathbf{x}\right)}{\sum\limits_{i=1}^{K} \exp\left(\boldsymbol{\theta}_i^t \cdot \mathbf{x}\right)} \tag{3.21}$$

Pentru clasificarea unui vector $\mathbf{x}$ dat la intrare, se calculează probabilitatea de apartenenţă la clasa k, $P(y=k|\mathbf{x};\mathbf{\Theta})$ pentru toţi k între 1 şi K şi se alege acel $k^{(max)}$ care realizează probabilitatea maximă:

$$k^{(max)} = \arg \max_{1 \leq k \leq K} P(y=k|\mathbf{x};\mathbf{\Theta}) \tag{3.22}$$

Predicţia făcută de model este că $\mathbf{x}$ aparţine clasei $k^{(max)}$. O strategie similară era folosită şi pentru regresia logistică binară.

Instruirea vizează determinarea acelei matrice $\mathbf{\Theta}$ pentru care modelul învaţă să recunoască cât mai bine datele din setul de instruire (eventual adăugându–se şi factor de regularizare). Mai clar, dacă un obiect $\mathbf{x}^{(j)}$ din setul de instruire $\mathcal{S}$ este de clasă $y^{(j)} = k$, atunci vrem ca valoarea $P\left(y=k|\mathbf{x}^{(j)};\mathbf{\Theta}\right)$ să fie cât mai aproape de 1.

3.3.4 Funcţia de cost

Ca şi în cazul funcţiei de eroare pentru regresia logistică, se va cuantifica în ce măsură diferă clasa actuală a unei intrări de predicţia dată de modelul $h_{\boldsymbol{\theta}}$. Introducem funcţia indicator $I(\cdot)$ care pentru o valoare logică returnează 1 dacă argmentul este adevărat şi 0 altfel:

$$I(valoare\_logica) = \begin{cases} 1 & \text{dacă } valoare\_logica = adevarat \\ 0 & \text{dacă } valoare\_logica = fals \end{cases} \tag{3.23}$$

Funcţia de eroare pentru regresia logistică multinomială se defineşte ca:

$$J(\mathbf{\Theta}) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} I\left(y^{(i)}=k\right) \cdot \ln \frac{\exp\left(\boldsymbol{\theta}_k^t \mathbf{x}^{(i)}\right)}{\sum\limits_{j=1}^{K} \exp\left(\boldsymbol{\theta}_j^t \mathbf{x}^{(i)}\right)}\right] \tag{3.24}$$

$$= -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} I\left(y^{(i)}=k\right) \cdot \ln P\left(y^{(i)}=k|\mathbf{x}^{(i)};\mathbf{\Theta}\right)\right] \tag{3.25}$$

Secţiunea 3.4 conţine sugestii pentru calcularea eficientă a funcţiei de eroare.

3.3.5 Calcularea gradientului

Determinarea lui $\boldsymbol{\theta}^{(min)} = \arg \min_{\boldsymbol{\theta} \in \mathbb{R}^{n+1}} J(\boldsymbol{\theta})$ se face prin căutarea după direcţia gradientului. Formula gradientului este, pentru $1 \le k \le K$:

$$\nabla_{\boldsymbol{\theta}_k} J(\boldsymbol{\Theta}) = -\frac{1}{m} \sum_{i=1}^{m} \left[\mathbf{x}^{(i)} \left(I(y^{(i)} = k) - P(y^{(i)} = k | \mathbf{x}^{(i)}; \boldsymbol{\Theta}) \right) \right] \tag{3.26}$$

pe care o demonstrăm în cele ce urmează. Având în vedere că gradientul e operator liniar, calculăm gradientul pentru un termen de cost corespunzător unei perechi de antrenare $\left(\mathbf{x}^{(i)}, y^{(i)}\right)$:

$$J_i(\boldsymbol{\Theta}) = -\sum_{l=1}^{K} I\left(y^{(i)} = l\right) \cdot \ln \frac{\exp\left(\boldsymbol{\theta}_l^t \mathbf{x}^{(i)}\right)}{\sum_{j=1}^{K} \exp\left(\boldsymbol{\theta}_j^t \mathbf{x}^{(i)}\right)} = \tag{3.27}$$

$$= -\sum_{l=1}^{K} I\left(y^{(i)} = l\right) \cdot \ln\left(\exp\left(\boldsymbol{\theta}_l^t \mathbf{x}^{(i)}\right)\right) + \tag{3.28}$$

$$+ \sum_{l=1}^{K} I\left(y^{(i)} = l\right) \cdot \ln\left(\sum_{j=1}^{K} \exp\left(\boldsymbol{\theta}_j^t \mathbf{x}^{(i)}\right)\right) = \tag{3.29}$$

$$= -\sum_{l=1}^{K} \left[I\left(y^{(i)} = l\right) \cdot \boldsymbol{\theta}_l^t \mathbf{x}^{(i)} \right] + \tag{3.30}$$

$$+ \sum_{l=1}^{K} I\left(y^{(i)} = l\right) \cdot \ln\left(\sum_{j=1}^{K} \exp\left(\boldsymbol{\theta}_j^t \mathbf{x}^{(i)}\right)\right) \tag{3.31}$$

În suma din linia (3.31), termenul cu logaritm nu depinde de indicele de însumare l, deci poate fi scos în faţa sumei; mai departe, pentru suma rămasă, pentru exact un indice l avem $I\left(y^{(i)} = l\right) = 1$ şi în rest indicatorii sunt 0, deci toată suma e 1. $J_i(\boldsymbol{\Theta})$ devine:

$$J_i(\boldsymbol{\Theta}) = -\sum_{l=1}^{K} \left[I\left(y^{(i)} = l\right) \cdot \boldsymbol{\theta}_l^t \mathbf{x}^{(i)} \right] + \ln\left(\sum_{j=1}^{K} \exp\left(\boldsymbol{\theta}_j^t \mathbf{x}^{(i)}\right)\right) \tag{3.32}$$

Aplicând gradientul, obţinem:

$$
\begin{aligned}
\nabla_{\boldsymbol{\theta}_k} J_i(\boldsymbol{\Theta}) &= -\nabla_{\boldsymbol{\theta}_k} \sum_{l=1}^{K} \left[I\left(y^{(i)} = l\right) \cdot \boldsymbol{\theta}_l^t \mathbf{x}^{(i)} \right] + \nabla_{\boldsymbol{\theta}_k} \ln \left(\sum_{j=1}^{K} \exp\left(\boldsymbol{\theta}_j^t \mathbf{x}^{(i)}\right) \right) \\
&= -I\left(y^{(i)} = k\right) \mathbf{x}^{(i)} + \frac{\nabla_{\boldsymbol{\theta}_k} \sum\limits_{j=1}^{K} \exp\left(\boldsymbol{\theta}_j^t \mathbf{x}^{(i)}\right)}{\sum\limits_{j=1}^{K} \exp\left(\boldsymbol{\theta}_j^t \mathbf{x}^{(i)}\right)} && (3.33) \\
&= -I\left(y^{(i)} = k\right) \mathbf{x}^{(i)} + \frac{\mathbf{x}^{(i)} \exp\left(\boldsymbol{\theta}_k^t \mathbf{x}^{(i)}\right)}{\sum\limits_{j=1}^{K} \exp\left(\boldsymbol{\theta}_j^t \mathbf{x}^{(i)}\right)} && (3.34) \\
&= -I\left(y^{(i)} = k\right) \mathbf{x}^{(i)} + \mathbf{x}^{(i)} P(y^{(i)} = k | \mathbf{x}^{(i)}; \boldsymbol{\Theta}) && (3.35) \\
&= -\mathbf{x}^{(i)} \left(I\left(y^{(i)} = k\right) - P(y^{(i)} = k | \mathbf{x}^{(i)}; \boldsymbol{\Theta}) \right) && (3.36)
\end{aligned}
$$

Prin calculul mediilor termenilor $\nabla_{\boldsymbol{\theta}_k} J_i(\boldsymbol{\Theta})$ pentru toti i se obţine formula gradientului total $\nabla_{\boldsymbol{\theta}_k} J(\boldsymbol{\Theta})$ din ecuaţia (3.26).

3.3.6 Algoritmul de instruire

Modificarea vectorului $\boldsymbol{\theta}_k$ este:

$$
\begin{aligned}
\boldsymbol{\theta}_k &= \boldsymbol{\theta}_k - \alpha \cdot \nabla_{\boldsymbol{\theta}_k} J(\boldsymbol{\Theta}) && (3.37) \\
&= \boldsymbol{\theta}_k + \frac{\alpha}{m} \sum_{i=1}^{m} \left[\mathbf{x}^{(i)} \left(I(y^{(i)} = k) - P(y^{(i)} = k | \mathbf{x}^{(i)}; \boldsymbol{\Theta}) \right) \right] && (3.38)
\end{aligned}
$$

pentru toţi $1 \leq k \leq K$. Modificarea se face iterativ, până când matricea $\boldsymbol{\Theta}$ se stabilizează[7], funcţia de eroare evoluează prea puţin[8], sau se atinge un număr maxim de iteraţii.

Algoritmul de instruire are forma:

1. Setează componentele lui $\boldsymbol{\Theta}$ la valori iniţiale aleatoare sau zero;
2. `repeta{`

   se fac atribuirile următoare, simultan pentru $1 \leq k \leq K$:

$$
\boldsymbol{\theta}_k := \boldsymbol{\theta}_k \frac{\alpha}{m} \sum_{i=1}^{m} \left[\mathbf{x}^{(i)} \left(I(y^{(i)} = k) - P(y^{(i)} = k | \mathbf{x}^{(i)}; \boldsymbol{\Theta}) \right) \right] \qquad (3.39)
$$

[7]Respectiv: norma diferenţei dintre două versiuni succesive ale matricei $\boldsymbol{\Theta}$ devine mai mică decât un prag ε dat, mic şi pozitiv. Drept normă se alege de regulă norma Frobenius: $\|\mathbf{A}\|_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} |a_{ij}^2|}$, matricea $\mathbf{A}$ fiind de componente $(a_{ij})_{1 \leq i \leq m, 1 \leq j \leq n}$.

[8]Mai clar: $|J(\boldsymbol{\Theta}_t) - J(\boldsymbol{\Theta}_{t-1})| < \varepsilon$, pentru un ε dat, pozitiv şi mic.

```
} pana la convergenta
```

Spre deosebire de regresia liniară, nu există o variantă algebrică de determinare a valorilor din matricea $\mathbf{\Theta}$.

3.3.7 Regularizare

În practică se preferă penalizarea valorilor absolute mari ale parametrilor θ_{li} $(1 \le l \le K, 1 \le i \le n)$; nu se penalizează ponderile de termeni liberi θ_{l0} $(1 \le l \le K)$. Se adaugă penalizări pentru pătratele valorilor ponderilor θ_{li} şi funcţia de eroare se modifică astfel:

$$J(\mathbf{\Theta}) = -\frac{1}{m}\left[\sum_{j=1}^{m}\sum_{l=1}^{K} I(y^{(j)} = l) \cdot \ln \frac{\exp\left(\boldsymbol{\theta}_l^t \mathbf{x}^{(j)}\right)}{\sum_{i=1}^{K} \exp\left(\boldsymbol{\theta}_i^t \mathbf{x}^{(j)}\right)}\right] + \frac{\lambda}{2m}\sum_{l=1}^{K}\sum_{i=1}^{n}\theta_{li}^2 \tag{3.40}$$

Coeficientul λ este un hiperparametru care arată cât de mult contează regularizarea în cadrul funcţiei de eroare; evident, pentru $\lambda = 0$ nu avem regularizare, iar dacă λ se alege foarte mare, atunci funcţia de eroare cross–entropy este neglijată în favoarea micşorării valorii coeficienţilor θ_{li}, fără a da vreo importanţă prea mare nepotrivirilor între clasele estimate şi cele reale. Evident, trebuie realizat un echilibru între aceste două extreme.

Formula gradientului folosit în căutarea de tip gradient descent este:

$$\nabla_{\boldsymbol{\theta}_k} J(\mathbf{\Theta}) = -\frac{1}{m}\sum_{j=1}^{m}\left[\mathbf{x}^{(j)}\left(I(y^{(j)} = k) - P(y^{(j)} = k|\mathbf{x}^{(j)}; \mathbf{\Theta})\right)\right] + \frac{\lambda}{m} \cdot [0, \boldsymbol{\theta}_{k,1:}]^t \tag{3.41}$$

unde $[0, \boldsymbol{\theta}_{k,1:}]^t$ este vectorul coloană care are pe prima poziţie 0 (termenii liberi nu se regularizează) şi pe următoarele n poziţii valorile $\theta_{k1}, \dots, \theta_{kn}$.

Valoarea hiperparametrului λ se determină prin încercări repetate, k fold cross-validation, căutare aleatoare, optimizare Bayesiană, algoritmi genetici sau alte euristici de căutare.

3.4 Comentarii

Elementele din această secţiune aduc precizări asupra modelelor de regresie logistică, precum şi discuţii legate de eficientizarea calculelor.

3.4.1 Redundanţa parametrilor pentru regresia logistică multinomială

Pentru regresia logistică cu două clase a fost suficient un singur parametru $\boldsymbol{\theta}$. Pentru modelul de regresie logistică multinomială cu K clase se folosesc K vectori $\boldsymbol{\theta}$, o discrepanţă vizibilă. Discuţia care urmează arată că, într-adevăr, regresia logistică multinomială are redundanţă de parametri.

Pentru o dată de intrare $\mathbf{x}$, modelul estimează probabilitatea condiţionată ca:

$$P(y = k|\mathbf{x}; \boldsymbol{\Theta}) = \frac{\exp(\boldsymbol{\theta}_k^t \mathbf{x})}{\sum\limits_{j=1}^{K} \exp\left(\boldsymbol{\theta}_j^t \mathbf{x}\right)} \tag{3.42}$$

Dacă scădem din fiecare vector $\boldsymbol{\theta}_j$ un acelaşi vector $\boldsymbol{\psi}$ şi calculăm funcţia softmax, obţinem:

$$\frac{\exp\left((\boldsymbol{\theta}_k - \boldsymbol{\psi})^t \mathbf{x}\right)}{\sum\limits_{j=1}^{K} \exp\left((\boldsymbol{\theta}_j - \boldsymbol{\psi})^t \mathbf{x}\right)} = \frac{\exp(-\boldsymbol{\psi}^t \mathbf{x}) \exp\left(\boldsymbol{\theta}_k^t \mathbf{x}\right)}{\exp(-\boldsymbol{\psi}^t \mathbf{x}) \sum\limits_{j=1}^{K} \exp\left(\boldsymbol{\theta}_j^t \mathbf{x}\right)} = \tag{3.43}$$

$$= \frac{\exp(\boldsymbol{\theta}_k^t \mathbf{x})}{\sum\limits_{j=1}^{K} \exp\left(\boldsymbol{\theta}_j^t \mathbf{x}\right)} = P(y = k|\mathbf{x}; \boldsymbol{\Theta}) \tag{3.44}$$

deci dacă avem un set de vectori $\{\boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_K\}$ care minimizează funcţia de cost J, atunci aceeaşi valoare minimă poate fi atinsă cu $\{\boldsymbol{\theta}_1 - \boldsymbol{\psi}, \dots, \boldsymbol{\theta}_K - \boldsymbol{\psi}\}$ pentru orice vector $\boldsymbol{\psi}$, iar datorită ec. (3.44) predicţiile vor fi aceleaşi ca la setul de parametri originar. Dacă pe post de $\boldsymbol{\psi}$ se ia oricare din vectorii $\boldsymbol{\theta}^k$, de exemplu $\boldsymbol{\theta}^K$, atunci vectorul de parametri devine neredundant, $\{\boldsymbol{\theta}_1 - \boldsymbol{\theta}_K, \boldsymbol{\theta}_2 - \boldsymbol{\theta}_K, \dots, \mathbf{0}\}$ cu acelaşi comportament ca şi înainte. Acceptăm însă redundanţa, pentru simplitatea si simetria formulelor rezultate. Redundanţa este demonstrată mai jos pentru cazul $K = 2$.

O observaţie interesantă e: chiar şi în cazul cu redundanţă funcţia de eroare J rămâne încă funcţie convexă, cu o infinitate de valori optime egale; metoda gradient descent nu ajunge în optime locale.

3.4.2 Relaţia dintre cele două tipuri de regresii logistice

Pentru regresia logistică multinomială, e de aşteptat ca să regăsim regresia logistică binară pentru cazul $K = 2$. Într-adevăr, ecuaţia (3.20) devine:

$$h_\Theta(\mathbf{x}) = \frac{1}{\exp(\boldsymbol{\theta}_\mathbf{1}^t \mathbf{x}) + \exp(\boldsymbol{\theta}_\mathbf{2}^t \mathbf{x})} \begin{pmatrix} \exp(\boldsymbol{\theta}_\mathbf{1}^t \mathbf{x}) \\ \exp(\boldsymbol{\theta}_\mathbf{2}^t \mathbf{x}) \end{pmatrix} \tag{3.45}$$

$$= \frac{1}{\exp(\boldsymbol{\theta}_\mathbf{1}^t \mathbf{x}) + \exp(\boldsymbol{\theta}_\mathbf{2}^t \mathbf{x})} \frac{\exp(\boldsymbol{\theta}_\mathbf{1}^t \mathbf{x})}{\exp(\boldsymbol{\theta}_\mathbf{1}^t \mathbf{x})} \begin{pmatrix} \exp(\boldsymbol{\theta}_\mathbf{1}^t \mathbf{x}) \\ \exp(\boldsymbol{\theta}_\mathbf{2}^t \mathbf{x}) \end{pmatrix} \tag{3.46}$$

$$= \frac{1}{\exp\left((\boldsymbol{\theta}_\mathbf{1} - \boldsymbol{\theta}_\mathbf{1})^t \mathbf{x}\right) + \exp\left((\boldsymbol{\theta}_\mathbf{2} - \boldsymbol{\theta}_\mathbf{1})^t \mathbf{x}\right)} \begin{pmatrix} \exp((\boldsymbol{\theta}_\mathbf{1} - \boldsymbol{\theta}_\mathbf{1})^t \mathbf{x}) \\ \exp((\boldsymbol{\theta}_\mathbf{2} - \boldsymbol{\theta}_\mathbf{1})^t \mathbf{x}) \end{pmatrix} \tag{3.47}$$

Notând $\boldsymbol{\theta} = \boldsymbol{\theta_1} - \boldsymbol{\theta_2}$, avem:

$$h_\Theta(\mathbf{x}) = \frac{1}{1 + \exp\left(-\boldsymbol{\theta}^t \cdot \mathbf{x}\right)} \begin{pmatrix} 1 \\ \exp\left(-\boldsymbol{\theta}^t \cdot \mathbf{x}\right) \end{pmatrix} \tag{3.48}$$

$$= \begin{pmatrix} \frac{1}{1+\exp(-\boldsymbol{\theta}^t \cdot \mathbf{x})} \\ 1 - \frac{1}{1+\exp(-\boldsymbol{\theta}^t \cdot \mathbf{x})} \end{pmatrix} \tag{3.49}$$

$$= \begin{pmatrix} P(y = 1|\mathbf{x}; \boldsymbol{\theta}) \\ 1 - P(y = 1|\mathbf{x}; \boldsymbol{\theta}) \end{pmatrix} \tag{3.50}$$

Tot pentru $K = 2$, se verifică uşor că funcţia de eroare (3.25) devine cea din (3.12).

3.4.3 Calculul numeric al funcţiei softmax

La implementarea directă a funcţie softmax se poate întâmpla ca valoarea funcţiei exponenţiale să depăşească posibilităţile de reprezentare numerică, pentru valori z_l mari. Un truc de calcul frecvent aplicat în practică este următorul: în loc de a se calcula funcţia softmax precum în ecuaţia (3.18), se calculează prima dată valoarea maximă din $(z_1, \dots, z_K)^t$, fie ea M, apoi se calculează softmax pentru vectorul de valori $\mathbf{z}' = (z_1 - M, \dots, z_K - M)^t$. Avem că:

$$softmax(\mathbf{z}'; l) = \frac{\exp(z_l - M)}{\sum\limits_{k=1}^{K} \exp(z_k - M)} = \frac{\exp(z_l)/\exp(M)}{\sum\limits_{k=1}^{K} [\exp(z_k)/\exp(M)]} =$$

$$= \frac{\exp(z_l)}{\sum\limits_{k=1}^{K} \exp(z_k)} = softmax(\mathbf{z}; l) \tag{3.51}$$

Rezultatul e acelaşi, dar calculul se face cu exponenţi mai mici, $z_l - M > z_l$. Se evită astfel depăşirea de reprezentare (overflow).

3.4.4 Trucul "log sum exp"

Funcţia de eroare pentru regresia logistică multinomială are forma:

$$J(\boldsymbol{\Theta}) = -\frac{1}{m} \left[\sum_{i=1}^{m} \sum_{k=1}^{K} I\left(y^{(i)} = k\right) \cdot \ln \frac{\exp\left(\boldsymbol{\theta}_k^t \mathbf{x}^{(i)}\right)}{\sum\limits_{j=1}^{K} \exp\left(\boldsymbol{\theta}_j^t \mathbf{x}^{(i)}\right)} \right] \tag{3.52}$$

Termenul cu logaritm permite simplificări de calcule:

$$\ln \frac{\exp\left(\boldsymbol{\theta}_k^t \mathbf{x}^{(i)}\right)}{\sum\limits_{j=1}^{K} \exp\left(\boldsymbol{\theta}_j^t \mathbf{x}^{(i)}\right)} = \ln\left(\exp\left(\boldsymbol{\theta}_k^t \mathbf{x}^{(i)}\right)\right) - \ln\left(\sum_{j=1}^{K} \exp\left(\boldsymbol{\theta}_j^t \mathbf{x}^{(i)}\right)\right) \tag{3.53}$$

$$= \boldsymbol{\theta}_k^t \mathbf{x}^{(i)} - \ln\left(\sum_{j=1}^{K} \exp\left(\boldsymbol{\theta}_j^t \mathbf{x}^{(i)}\right)\right) \tag{3.54}$$

Pentru al doilea termen din (3.54) se foloseşte un truc asemănător celui din subsecţiunea precedentă, pentru a evita calcul de funcţie exponenţială cu exponenţi mari. Trucul "log sum exp" se bazează pe identitatea:

$$\ln\left(\sum_{j=1}^{K} \exp(z_i)\right) = a + \ln\left(\sum_{j=1}^{K} \exp(z_i - a)\right) \tag{3.55}$$

pentru un a oarecare, demonstrată în cele ce urmează:

$$\ln\left(\sum_{j=1}^{K} \exp(z_i)\right) = \ln\left(\sum_{j=1}^{K} \left(\exp(z_i)\exp(a)\exp(-a)\right)\right) \tag{3.56}$$

$$= \ln\left(\exp(a) \cdot \sum_{j=1}^{K} \left(\exp(z_i)\exp(-a)\right)\right) \tag{3.57}$$

$$= a + \ln\left(\sum_{j=1}^{K} \exp(z_i - a)\right) \tag{3.58}$$

Pentru a reduce din magnitudinea exponenţilor considerăm, în mod convenabil, $a = \max_i z_i$ şi reducem riscul de depăşire de reprezentare numerică (overflow).

Capitolul 4

Reţele neurale artificiale - fundamente

4.1 Încadrarea domeniului

Studiul reţelelor neurale artificiale a fost motivat de observaţia că un sistem biologic este mai performant decât calculatoarele şi programele existente la ora actuală într-o serie de sarcini precum recunoaşterea de imagini, regăsirea informaţiei, înţelegerea vorbirii. Acest lucru se întâmplă cu toate că neuronii biologici operează mult mai lent decât procesoarele actuale. Studiul reţelelor neurale artificiale are ca scop producerea unor sisteme care să obţină rezolvări pentru probleme de tipul celor de mai sus, dar şi altele de natură sintetică, pe baza experienţei acumulate din mediu. La ora actuală, reţelele neurale artificiale nu mai au o atât de mare afinitate cu biologia, fiind mai degrabă modele matematice utilizate pentru aproximare.

Definiţia următoare ([3]) ia în considerare abilitatea de adaptare pe baza experienţei:

Definiţia 4. *Sistemele neurale artificiale, sau reţelele neurale sunt sisteme celulare fizice care pot achiziţiona, stoca şi utiliza cunoaşterea experimentală.*

Natura neliniară, complexă şi cu grad mare de paralelism reprezintă diferenţe majore faţă de modelele de calcul actuale. O reţea neurală artificială modelează felul în care creierul biologic procesează semnalele. Modelele de reţele neurale artificiale sunt structurate ca nişte grafuri ale căror noduri sunt neuroni artificiali, iar legăturile dintre noduri au ponderi[1] - valori numerice - ajustabile printr-un proces de învăţare. Definiţia din [1] sumarizează acest fapt:

Definiţia 5. *O reţea neurală este un sistem de procesare masiv paralel-distribuit constând din unităţi de procesare simple, care au predispoziţie*

[1]În limba engleză: weights.

naturală pentru stocarea cunoştinţelor experimentale şi utilizarea lor. Este asemănătoare creierului în două aspecte:

1. *Cunoaşterea este achiziţionată de către reţea din mediu printr–un proces de învăţare;*

2. *Ponderile legăturilor dintre neuroni, cunoscute ca ponderi sinaptice sunt folosite pentru a reţine cunoaşterea dobândită.*

Procedura prin care se obţine adaptarea ponderilor din cadrul reţelei se numeţe *algoritm de învăţare*. Mai menţionăm că învăţarea poate duce la modificarea numărului de noduri din reţea (a se vedea de exemplu Fuzzy ARTMAP, capitolul 9), ceea ce în cadrul reţelelor neurale biologice are ca şi corespondent faptul că unii neuroni mor (efectul de retezare din reţele neurale) şi alţi neuroni se pot alătura celor existenţi pentru a sprijini învăţarea.

Numele sub care mai sunt cunoscute aceste reţele sunt: neurocalculatoare, reţele conecţioniste, sisteme adaptive stratificate, reţele cu auto-organizare, sisteme neuromorfice etc.

Există multe moduri în care pot fi privite aceste reţele neurale de către diferite categorii profesionale:

- oamenii de ştiinţă din domeniul neurobiologiei sunt interesaţi de modul în care reţelele neurale artificiale confirmă rezultatele sau modelele cunoscute pentru sisteme biologice; facem precizarea că transferul de cunoştinţe nu este doar dinspre biologie spre domeniul reţelelor neurale artificiale, ci şi invers: modele teoretice sunt confirmate de descoperirile biologice[2];

- fizicienii văd analogii între reţelele neurale şi sistemele dinamice neliniare pe care le studiază;

- matematicienii sunt interesaţi de potenţialul de modelare matematică pentru sisteme complexe;

- inginerii din domeniul electric le folosesc pentru procesarea de semnale;

- informaticienii sunt interesaţi de oportunităţile care apar in zonele de inteligenţă artificială, teorie computaţională, modelare şi simulare etc.

Beneficiile aduse de reţele neurale artificiale sunt:

1. neliniaritatea: un neuron artificial poate avea comportament liniar sau nu; caracteristica neliniară este importantă pentru cazul în care mecanismul care generează semnalul este neliniar - de exemplu semnalul de vorbire;

[2]Vezi de exemplu "Reinforcement learning through modulation of spike-timing-dependent synaptic plasticity", https://florian.io/papers/2007_Florian_Modulated_STDP.pdf.

2. detectarea de asocieri între intrări şi ieşiri: este cazul învăţării supervizate, în care antrenarea se face pe baza unor perechi de semnale, corespunzătoare intrărilor şi respectiv ieşirilor asociate. Se poate pleca de la un model care nu are cunoştinţe apriori despre domeniu şi pe baza datelor se învaţă asocierea.

3. adaptabilitate: reţelele neurale au capacitatea naturală de a–şi adapta ponderile în funcţie de semnalele provenite din mediu; mediul poate să fie nestaţionar, adică să sufere modificări în timp ale distribuţiei semnalelor;

4. rezistenţa la zgomot: o reţea neurală poate să accepte date care au imprecizie sau sunt afectate de zgomot; sunt raportate chiar situaţii în care adăugarea de zgomot la datele de antrenare îmbunătăţeşte calitatea învăţării.

4.2 Neuronul biologic

Este utilă o scurtă incursiune în biologie, pentru a înţelege modelarea ce se face pentru un neuron artificial.

Neuronul biologic este o celulă nervoasă elementară, elementul constructiv de bază pentru reţeaua neurală biologică. Neuronul are trei părţi principale: corpul celulei, numit şi *soma*, *axonul* şi *dendritele*. O schemă a unui neuron a fost dată în figura 1.1. Dendritele formează o arborescenţă prin care sunt primite impulsuri de la alţi neuroni. Axonul este un fir conductor lung, cu un capăt în corpul celulei iar cu celălalt ramificat, prin care se trimite semnal către dendritele altor axoni. Contactul dintre un axon şi o dendrită se cheamă sinapsă. Ca mediatori ai semnalului în sinapse se folosesc adrenalină, noradrenalină, acetilcolina, serotonina.

Neuronul este capabil să dea un răspuns pe baza intrărilor furnizate de către dendrite. Răspunsul acesta este generat dacă potenţialul membranei depăşeste un anumit prag de activare. Impulsurile care vin prin membrană sunt *excitatoare* dacă ele favorizează formarea semnalului de ieşire din neuron şi *inhibitoare* dacă inhibă răspunsul. Se face o agregare a semnalelor primite de celulă de-a lungul unei perioade de sumare latentă, luându-se în considerare şi apropierea în timp a semnalor excitatoare primite; nu se cere o sincronizare a semnalelor, iar valoarea de ieşire este de regulă văzută ca una binară: dacă suma semnalelor primite este mai mare decât pragul de activare (nu contează cu cât mai mare) se trimite semnal mai departe prin axon, către alţi neuroni; dacă este mai mic, atunci nu se declanşează semnal în axon.

După emiterea semnalului prin axon există o perioadă refractară, în care neuronul nu mai ia în considerare niciun semnal sosit, indiferent de gradul de intensitate. După scurgerea acestei perioade, neuronul este gata să proceseze noi semnale. Perioada refractară nu are neapărat aceeaşi durată pentru toate celulele. Timpul necesar acestui ciclu este de ordinul milisecundelor.

Facem precizarea că prezentarea făcută este o variantă simplificată; considerarea şi modelarea unor detalii duce la reţele neurale artificiale de diferite tipuri (generaţii).

4.3 Modele de neuroni artificiali

4.3.1 Modelul McCulloch–Pitts

Reprezintă prima definiţie formală a unui neuron artificial, ce pleacă de la descrierea neuronului biologic. Modelul a fost formulat în 1943 de neurobiologul si ciberneticianul Warren McCulloch şi respectiv logicianul Walter Pitts. Modelul este arătat în figura 4.1. Intrările x_i sunt 0 sau 1, $i = \overline{1,n}$, ponderile $w_i \in \{-1, 1\}$, T este pragul neuronului iar ieşirea o se calculează ca:

$$o^{k+1} = \begin{cases} 1 & \text{dacă } \sum\limits_{i=1}^{n} w_i x_i^k \geq T \\ 0 & \text{altfel} \end{cases} = I(\mathbf{w}^t \cdot \mathbf{x} \geq T) \tag{4.1}$$

unde $I(\cdot)$ este funcţia indicator dată în ecuaţia (3.23) pagina 42, $\mathbf{w} = (w_1, w_2, \dots, w_n)^t$ şi $\mathbf{x} = (x_1, x_2, \dots, x_n)^t$ sunt respectiv vectorul de ponderi şi de valori de intrare, iar indicele superior k este momentul de timp, $k = 0, 1, \dots$. Ponderile $w_i = 1$ reprezintă sinapsele excitatoare, cele cu valoare -1 sunt inhibitoare. Cu toate că modelul este simplu, el poate fi folosit pentru implementarea operaţiilor logice `not`, `and` şi `or`, dacă ponderile şi pragul sunt setate corespunzător.

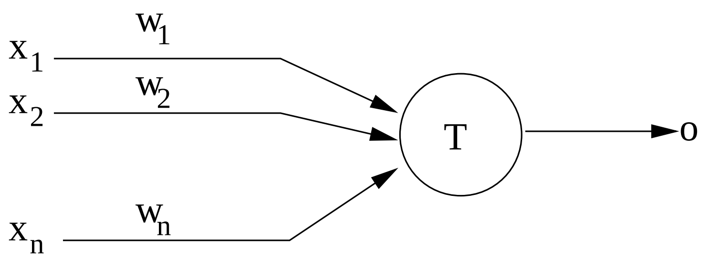


Figura 4.1: Modelul McCulloch-Pitts pentru neuron artificial.

4.3.2 Modelarea neuronului pentru sisteme neurale artificiale

Modelul McCulloch-Pitts este o viziune simplificată: permite doar intrări, ieşiri şi ponderi binare, operează în timp discret, presupune sincronizarea intrărilor (toate intrările trebuie să sosească în acelaşi timp), ponderile şi pragul sunt fixe. Ca atare, se propune modelul dat în figura 4.2 cu următoarele precizări:

1. fluxul semnalului de intrare x_i este unidirecţional
2. valoarea de ieşire a neuronului este:

$$o = f(\mathbf{w}^t \cdot \mathbf{x}) = f\left(\sum_{i=1}^{n} w_i x_i\right) \tag{4.2}$$

   unde $\mathbf{w}$ şi $\mathbf{x}$ au fost daţi mai sus.

3. valoarea produsului scalar $\mathbf{w}^t \cdot \mathbf{x}$ se notează cu *net* şi se numeşte valoare de activare a neuronului;
4. funcţia f se numeşte *funcţie de activare*; se vor da mai multe variante.

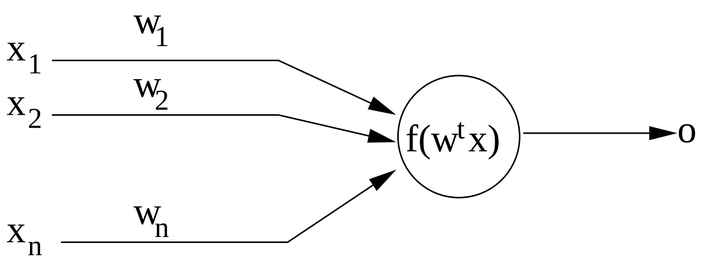


Figura 4.2: Model general de neuron

Se remarcă lipsa pragului T din ecuaţiile de mai sus; de fapt, se poate considera că neuronul are doar $n-1$ intrări sinaptice iar valoarea a n-a este $x_n = -1$, permanent, având ponderea asociată $w_n = T$. Funcţiile de activare $f(\cdot)$ larg folosite sunt

$$f_1(net) = \frac{2}{1 + e^{-\lambda \cdot net}} - 1, \lambda > 0 \tag{4.3}$$

cu graficul dat în figura 4.3 şi

$$f_2(net) = sgn(net) = \begin{cases} +1, & \text{dacă } net > 0 \\ -1, & \text{dacă } net < 0 \end{cases} \tag{4.4}$$

Se observă că $\lambda/4 = f_1'(0)$, deci λ este proporţională cu panta tangentei la graficul lui f_1 în origine. Pentru $\lambda \to \infty$ avem că f_1 tinde către f_2. Datorită alurii funcţiei f_1, amintind de forma literei S, ea se mai numeşte şi sigmoidă. Se mai poate folosi drept funcţie de activare tangenta hiperbolică:

$$f_3(x) = \tanh(x) = \frac{\sinh(x)}{\cosh(x)} = \frac{\frac{\exp(x)-\exp(-x)}{2}}{\frac{\exp(x)+\exp(-x)}{2}} = \frac{\exp(2x) - 1}{\exp(2x) + 1} \tag{4.5}$$

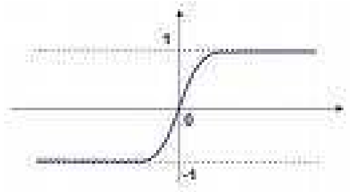


Figura 4.3: Funcţia sigmoidă dată de ecuaţia 4.3 pentru λ fixat.

Deoarece funcţiile date iau valori între -1 şi +1 se mai numesc bipolare, iar aspectul continuu sau discontinuu le dă denumirea de *bipolară continuă* şi respectiv *bipolară discretă*. Funcţia bipolară poate fi redusă la formă unipolară, mărginită de 0 şi 1:

$$f_4(net) = \sigma_\lambda(net) = \frac{1}{1+e^{-\lambda \cdot net}}, \lambda > 0 \tag{4.6}$$

(sigmoida logistică) şi

$$f_5(net) = \begin{cases} +1, & \text{dacă } net > 0 \\ 0, & \text{dacă } net < 0 \end{cases} \tag{4.7}$$

Drept funcţie de activare se poate folosi de fapt orice funcţie care este monoton nedescrescătoare, mărginită şi preferabil derivabilă.

Ieşirile pot fi binare sau continue, bipolare sau unipolare. Pentru m neuroni, mulţimea valorilor de ieşire este:

- $(-1, 1)^m$ pentru funcţie de activare bipolară continuă
- $(0, 1)^m$ pentru funcţie de activare unipolară continuă
- $\{-1, 1\}^m$ pentru funcţie de activare bipolară discretă
- $\{0, 1\}^m$ pentru funcţie de activare unipolară discretă

4.4 Modele de reţea neurală artificială

4.4.1 Reţea cu propagare înainte

Vom considera o arhitectură de reţea cu propagare înainte[3] cu n intrări şi m neuroni de ieşire, precum în figura 4.4. Intrările şi ieşirile sunt respectiv:

$$\begin{aligned} \mathbf{x} &= (x_1, x_2, \ldots, x_n)^t \\ \mathbf{o} &= (o_1, o_2, \ldots, o_m)^t \end{aligned} \tag{4.8}$$

Dacă considerăm vectorul de ponderi $\mathbf{w}_i$ care leagă neuronul de ieşire i cu

[3] În original: feedforward network.

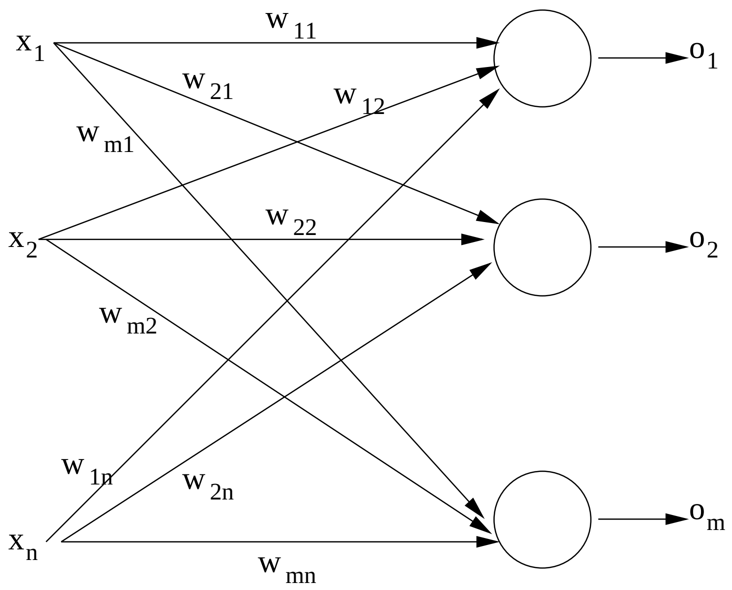


Figura 4.4: Model de reţea neurală artificială

toate intrarile, $\mathbf{w}_i = (w_{i1}, w_{i2}, \dots, w_{in})^t$, atunci valoarea de activare pentru neuronul i este

$$net_i = \mathbf{w}_i^t\mathbf{x} = \sum_{j=1}^{n} w_{ij}x_j, 1 \leq i \leq m \tag{4.9}$$

Valoarea de ieşire o_i pentru fiecare neuron este $o_i = f(net_i) = f(\mathbf{w}_i^t\mathbf{x})$.

Putem nota cu $\mathbf{W}$ matricea ponderilor dintre neuroni:

$$\mathbf{W} = \begin{pmatrix} w_{11} & w_{12} & \cdots & w_{1n} \\ w_{21} & w_{22} & \cdots & w_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m1} & w_{m2} & \cdots & w_{mn} \end{pmatrix} \tag{4.10}$$

şi introducem operatorul matricial $\Gamma(\cdot)$ definit ca

$$\Gamma \begin{pmatrix} net_1 \\ net_2 \\ \vdots \\ net_m \end{pmatrix} = \begin{pmatrix} f(net_1) \\ f(net_2) \\ \vdots \\ f(net_m) \end{pmatrix} \tag{4.11}$$

ceea ce ne permite să scriem ieşirea reţelei ca fiind:

$$\mathbf{o} = \Gamma(\mathbf{W} \cdot \mathbf{x}) \tag{4.12}$$

Valorile de intrare $\mathbf{x}$ şi cele de ieşire $\mathbf{o}$ se numesc pattern-uri de intrare şi respectiv de ieşire. Reţeaua acţionează instantaneu, adică de îndată ce intrarea este furnizată, reţeaua dă şi valoarea de ieşire asociată. Dacă se consideră momentul de timp t, atunci putem rescrie 4.12 ca:

$$\mathbf{o}(t) = \Gamma(\mathbf{W} \cdot \mathbf{x}(t)) \tag{4.13}$$

4.4.2 Reţele cu conexiune inversă

La reţeaua prezentată anterior putem adăuga nişte conexiuni care să facă legătura de la ieşiri la intrări. Reţeaua nou obţinută este denumită "cu conexiune inversă"[4]; o reprezentare este dată în figura 4.5. În felul acesta, ieşirile controlează intrările. Mai mult, valorile $o(t)$ controlează valorile $o(t+\Delta)$. Δ reprezintă aici perioada refractară a neuronului. Ieşirea este dată de ecuaţia:

$$\mathbf{o}(t+\Delta) = \Gamma(\mathbf{W}\mathbf{o}(t)) \tag{4.14}$$

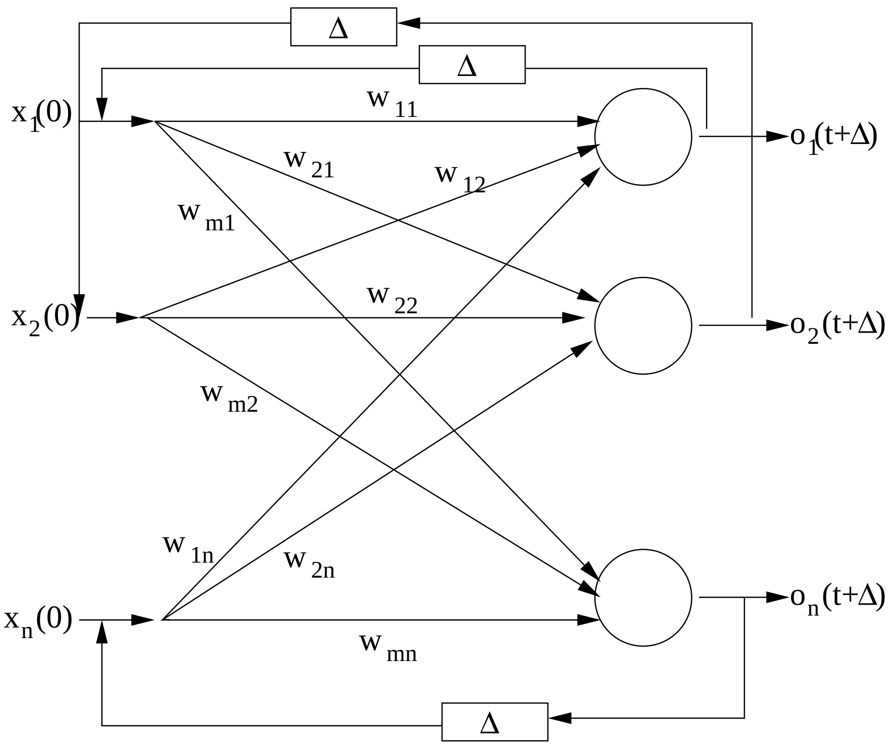


Figura 4.5: Reţea cu conexiune inversă.

Intrarea $\mathbf{x}$ este necesară doar la început ($t = 0$), după care sistemul se auto-întreţine. Dacă considerăm timpul ca valoare discretă şi urmărim sistemul la momentele 0, Δ, 2Δ, $\ldots$, $k\Delta$, $\ldots$ atunci sistemul se numeşte discret. Putem lua convenabil $\Delta = 1$ şi atunci avem:

$$\mathbf{o}^{k+1} = \Gamma(\mathbf{W}\mathbf{o}^k), k = 1, 2, \ldots \tag{4.15}$$

[4]În limba engleză: feedback network.

sau

$$\mathbf{o}^{k+1} = \Gamma\left(\mathbf{W}\Gamma\left(\cdots\Gamma(\mathbf{W}\mathbf{x}^0)\cdots\right)\right) \tag{4.16}$$

Şirul de valori $\mathbf{o}^1$, $\mathbf{o}^2$, ... reprezintă stările succesive ale reţelei, care în acest caz este văzut ca un sistem dinamic. Este posibil ca de la un moment dat $\mathbf{o}^k = \mathbf{o}^{k+1} = \ldots$, adică $\mathbf{o}^k$ să fie un atractor, iar reţeaua se stabilizează. Mai general, un atractor poate să fie o mulţime finită de valori. Un exemplu de astfel de reţea neurală este memoria asociativă bidirecţională.

4.5 Învăţarea ca problemă de aproximare

În urma procesului de învăţare nu putem obţine în toate cazurile reproducerea perfectă a ceea ce s–a învăţat. Se poate obţine o aproximare a unei funcţii $h(\cdot)$ printr-o funcţie $H(\mathbf{w}, \cdot)$ unde $\mathbf{w} = (w_1, w_2, \ldots w_m)^t$ iar argumentul notat "$\cdot$" este un vector $\mathbf{x} = (x_1, x_2, \ldots, x_n)^t$. Problema este de a găsi vectorul $\mathbf{w}$ care dă cea mai bună aproximare pentru un set de antrenare $\{(\mathbf{x}^1, h(\mathbf{x}^1)), \ldots, (\mathbf{x}^p, h(\mathbf{x}^p))\}$. Primul pas este alegerea funcţiei aproximante $H(\cdot, \cdot)$, apoi un proces de învăţare este folosit pentru a determina o valoare bună pentru vectorul $\mathbf{w}$. Altfel zis, se caută un vector $\mathbf{w}^*$ pentru care

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \sum_{k=1}^{p} \rho\left(H(\mathbf{w}, \mathbf{x}^k), h(\mathbf{x}^k)\right) \tag{4.17}$$

unde ρ este o funcţie distanţă care măsoară calitatea aproximării. Învăţarea este procesul de găsire a unui bun aproximant.

Deşi formularea este simplă, există două dificultăţi în rezolvarea generală a acestei probleme de aproximare:

1. o valoare potrivită pentru m poate fi greu de determinat; atunci când aproximarea se face prin reţele neurale de tip feedforward cu un strat ascuns (vezi curs 6), valoarea lui m este numărul de neuroni din stratul ascuns;

2. determinarea efectivă a lui $\mathbf{w}^*$ cunoaşte rezolvări pentru câteva clase de funcţii H şi ρ – a se vedea teoria cercetărilor operaţionale – dar o metodă eficientă pentru toate cazurile nu este cunoscută[5]; putem vorbi de probleme de programare liniară sau de programare pătratică, dar sunt multe alte situaţii care nu au o rezolvare teoretică cunoscută. O abordare este folosirea unor metode de căutare euristică – metodele "hill-climbing", "simulated annealing", algoritmi genetici, dar aceştia nu garantează obţinerea minimului absolut.

[5] A se vedea http://en.wikipedia.org/wiki/No_free_lunch_in_search_and_optimization.

4.6 Reguli de învăţare

În această secţiune vom privi neuronul artificial ca pe o entitate adaptivă, pentru care ponderile se pot modifica pe baza unui proces de învăţare. Ponderile se modifică în funcţie de:

- valoarea actuală a ponderilor, $\mathbf{w}$ sau $\mathbf{W}$;
- semnalul de intrare $\mathbf{x}$;
- ieşirea rezultată $\mathbf{o}$;
- (opţional) ieşirea furnizată de un "profesor", în cazul învăţării supervizate, numită şi ieşirea dorită, $\mathbf{d}$.

Putem presupune că intrarea x_n are valoarea fixă -1, pentru a permite includerea parametrului prag. Putem considera că sunt n neuroni de intrare şi m de ieşire, iar vectorul ponderilor care leagă al i-lea neuron de ieşire de toţi neuronii de intrare este $\mathbf{w}_i = (w_{i1}, w_{i2}, \ldots, w_{in})^t$.

Regula de învăţare generală este că ponderile $\mathbf{w}_i$ variază proporţional cu produsul dintre intrarea $\mathbf{x}$ şi semnalul de învăţare r. r este o funcţie care ia în considerare trei din valorile date mai sus, deci

$$r = r(\mathbf{w}_i, \mathbf{x}, d_i) \tag{4.18}$$

unde d_i este eventuala valoare de ieşire ce corespunde neuronului de ieşire i. Mai precis, avem

$$\Delta\mathbf{w}_i(t) = \alpha \cdot r(\mathbf{w}_i(t), \mathbf{x}(t), d_i(t)) \cdot \mathbf{x}(t) \tag{4.19}$$

unde $\alpha > 0$ este rata de învăţare. Expresia (4.19) dă modul în care se modifică ponderile de la un moment de timp la următorul:

$$\mathbf{w}_i(t+1) = \mathbf{w}_i(t) + \alpha \cdot r(\mathbf{w}_i(t), \mathbf{x}(t), d_i(t)) \cdot \mathbf{x}(t) \tag{4.20}$$

Pentru cazul discret se foloseşte scrierea:

$$\mathbf{w}_i^{k+1} = \mathbf{w}_i^k + \alpha \cdot r(\mathbf{w}_i^k, \mathbf{x}^k, d_i^k) \cdot \mathbf{x}^k, k = 0, 1, \ldots \tag{4.21}$$

iar pentru cazul continuu se scrie ecuaţia diferenţială

$$\frac{\mathrm{d}\mathbf{w}_i(t)}{\mathrm{d}t} = \alpha \cdot r(\mathbf{w}_i(t), \mathbf{x}(t), d_i(t))\mathbf{x}(t) \tag{4.22}$$

Pe baza formei funcţiei r avem variantele de învăţare menţionate mai jos.

4.6.1 Regula de învăţare Hebbiană

Este o regulă de învăţare nesupervizată formulată de Donald Hebb în 1949 astfel:

> When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased.

Matematic, se scrie:

$$r = f(\mathbf{w}_i^t\mathbf{x}) \tag{4.23}$$

deci modificarea ponderilor devine

$$\Delta\mathbf{w}_i = \alpha f(\mathbf{w}_i^t\mathbf{x})\mathbf{x} \tag{4.24}$$

ceea ce pe componente se scrie

$$\Delta w_{ij} = \alpha f(\mathbf{w}_i^t\mathbf{x})x_j \tag{4.25}$$

sau în funcţie de ieşirea neuronului i

$$\Delta w_{ij} = \alpha o_i x_j \tag{4.26}$$

Ponderile sunt iniţializate cu valori aleatoare mici, în jurul lui 0. Conform formulelor date, putem avea o creştere a ponderii w_{ij} dacă produsul $o_i x_j$ este pozitiv şi o scădere în caz contrar. Se arată uşor că intrările prezentate frecvent au o influenţă mai mare asupra ponderilor şi vor produce o valoare de ieşire mare.

Regula trebuie înţeleasă ca un principiu, existând la ora actuală variaţiuni pe această temă.

Exemplu numeric: [3] pag 61.

4.6.2 Regula de învăţare a perceptronului

Este folosită pentru învăţare supervizată; regula a fost formulată de către Rosenblatt în 1958. Semnalul de învăţare este diferenţa între valoarea dorită şi cea obţinută:

$$r = d_i - o_i \tag{4.27}$$

unde $o_i = sgn(\mathbf{w}_i^t\mathbf{x})$, iar d_i este răspunsul dorit, furnizat de "profesor". Modificarea ponderilor este deci

$$\Delta\mathbf{w}_i = \alpha\left(d_i - sgn(\mathbf{w}_i^t\mathbf{x})\right)\mathbf{x} \tag{4.28}$$

Formula se aplică pentru cazul în care ieşirile sunt bipolare binare. Modificarea în ponderi apare doar dacă ieşirea furnizată de neuronul de ieşire i

diferă de valoarea dorită — cunoscută aprioric. Explicitând, se poate vedea că modificarea de pondere în cazul neconcordanţei ieşirii cu valoarea dorită este

$$\Delta \mathbf{w}_i = \pm 2\alpha \mathbf{x} \tag{4.29}$$

Ponderile pot fi iniţializate cu orice valoare.

Exemplu: [3] pag 65.

4.6.3 Regula de învăţare delta

Prezenta regulă se foloseşte pentru cazul învăţării supervizate cu funcţie de activare derivabilă; a fost introdusă de McClelland şi Rumelhart în 1986. Semnalul de învăţare se cheamă în acest context "delta" şi are forma:

$$r = (d_i - f(\mathbf{w}_i^t \mathbf{x}))f'(\mathbf{w}_i^t \mathbf{x}) \tag{4.30}$$

Motivaţia prezenţei derivatei în formulă este dată de minimizarea erorii pătratice:

$$E = \frac{1}{2}(d_i - o_i)^2 = \frac{1}{2}\left(d_i - f(\mathbf{w}_i^t \mathbf{x})\right)^2 \tag{4.31}$$

Tehnica de reducere a valorii funcţiei constă în mişcarea în sens opus gradientului ∇E

$$\nabla E = -(d_i - o_i)f'(\mathbf{w}_i^t \mathbf{x})\mathbf{x} \tag{4.32}$$

Componentele gradientului sunt

$$\frac{\partial E}{\partial w_{ij}} = -(d_i - o_i)f'(\mathbf{w}_i^t \mathbf{x})x_j, \forall j = \overline{1, n} \tag{4.33}$$

Modificarea ponderilor se face astfel:

$$\Delta \mathbf{w}_i = -\alpha \nabla E \tag{4.34}$$

unde η este o constantă pozitivă. O altă scriere este:

$$\Delta \mathbf{w}_i = \alpha(d_i - o_i)f'(net_i)\mathbf{x} \tag{4.35}$$

Regula poate fi generalizată pentru reţele cu mai multe straturi.

Exemplu: [3] pag 68.

4.6.4 Regula de învăţare Widrow-Hoff

A fost enunţată în 1962 şi se aplică pentru învăţarea supervizată. Regula foloseşte ca funcţie de activare funcţia identică $f(x) = x$ şi minimizează pătratul diferenţei dintre ieşirea dorită d_i şi activarea net_i:

$$r = d_i - \mathbf{w}_i^t \mathbf{x} \tag{4.36}$$

deci ajustarea de ponderi se face cu

$$\Delta \mathbf{w}_i = \alpha(d_i - \mathbf{w}_i^t \mathbf{x})\mathbf{x} \tag{4.37}$$

Regula Widrow-Hoff este o formă particulară a regulii delta şi mai este cunoscută sub numele de regula celor mai mici pătrate[6]. Ponderile sunt iniţializate cu orice valori.

4.6.5 Regula de învăţare prin corelaţie

Se obţine prin considerarea lui $r = d_i$. Ponderile se modifică cu valoarea

$$\Delta \mathbf{w}_i = \alpha d_i \mathbf{x} \tag{4.38}$$

Ponderile sunt iniţializate cu zero.

4.6.6 Regula "câştigătorul ia tot"

Regula "winner-takes-all" este un exemplu de învăţare competitivă şi e folosită pentru învăţarea în mod nesupervizat a proprietăţilor statistice ale datelor. Învăţarea se bazează pe premisa că din toţi neuronii de ieşire unul (fie el de indice k) dă răspunsul maxim pentru o intrare $\mathbf{x}$. Ponderea aferentă acestui vector va fi modificată astfel:

$$\Delta \mathbf{w}_i = \alpha(\mathbf{x} - \mathbf{w}_i) \tag{4.39}$$

unde $\alpha > 0$ este o valoare mică, de regulă descrescătoare pe măsură ce procesul de învăţare continuă. Indicele i este ales deci

$$i = \arg\max_k \left(\mathbf{w}_k^t \mathbf{x}\right) \tag{4.40}$$

După ajustare, ponderile tind să estimeze mai bine patternul de intrare. Ponderile sunt iniţializate cu valori aleatoare şi lungimile lor sunt apoi normalizate: $\|w_i\| = const$, $\forall i$.

[6] Least Mean Square (LMS).

Capitolul 5

Perceptronul liniar

5.1 Motivaţie, definiţii, notaţii

Scopul cursului este de a prezenta perceptronul liniar - cel mai simplu tip de neuron, capabil să înveţe să separe două mulţimi (clase) de puncte care sunt liniar separabile.

Pentru perceptronul liniar se foloseşte instruire supervizată, modelul rezultat putând fi folosit pentru clasificare.

Definiţia 6. *(Clase liniar separabile) Mulţimile C_1 şi C_2 de puncte din spaţiul $\mathbb{R}^n$ se numesc liniar separabile dacă există o funcţie $y : \mathbb{R}^n \to \mathbb{R}$ de forma:*

$$y(\mathbf{x}) = \sum_{i=1}^{n} w_i \cdot x_i + b \tag{5.1}$$

astfel încât:

$$\begin{cases} y(\mathbf{x}) > 0 \text{ pentru } \mathbf{x} \in C_1 \\ y(\mathbf{x}) < 0 \text{ pentru } \mathbf{x} \in C_2 \end{cases} \tag{5.2}$$

În condiţiile de mai sus, numim clasa C_1 clasă pozitivă, iar C_2 clasă negativă.

Plecând de la un set de instruire format din mulţimile C_1, C_2 despre care se ştie că sunt liniar separabile – dar pentru care funcţia $y(\cdot)$ nu se cunoaşte – perceptronul liniar determină coeficienţii $b, w_1, \dots, w_n$ pentru care condiţiile din (5.2) sunt îndeplinite.

Ecuaţia $y(\mathbf{x}) = 0$, precum şi mulţimile C_1, C_2 pe care y le separă au interpretări geometrice simple. Punctele $\mathbf{x} \in \mathbb{R}^n$ pentru care $y(\mathbf{x}) = 0$ formează o varietate liniară – notată cu $\mathcal{S}$ – de dimensiune $n-1$; dacă $b = 0$ atunci $y(\mathbf{x}) = 0$ este un hiperplan (subspaţiu afin de dimensiune $n-1$ şi deci trecând prin origine); prin abuz de limbaj, în literatură se foloseşte tot denumirea de hiperplan pentru suprafaţa $y(\mathbf{x}) = 0$ chiar dacă $b \neq 0$.

Pentru cazul $n = 2$, $w_1 > 0$, $w_2 > 0$, $b < 0$ a se vedea reprezentarea din figura 5.1.

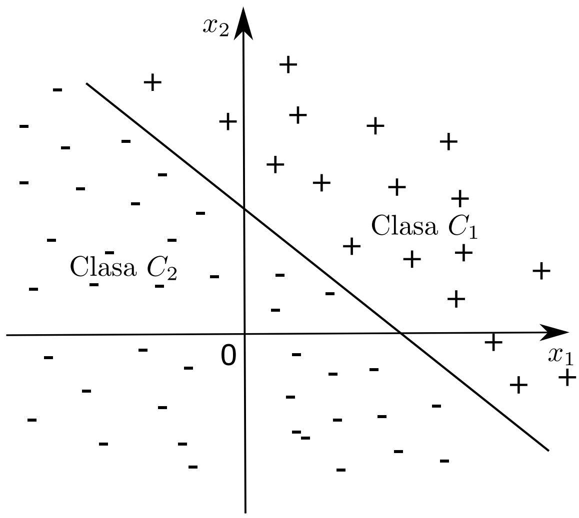


Figura 5.1: Mulţimi separabile liniar şi suprafaţă de separare liniară $y(\mathbf{x}) = w_1x_1 + w_2x_2 + b$, $w_1 > 0$, $w_2 > 0$, $b < 0$. Clasa C_1 e clasă pozitivă, clasa C_2 e clasă negativă

Suprafaţa liniară $y(\mathbf{x}) = 0$ separă planul în două submulţimi – în cazul $n = 2$, în două semiplane. Oricum am alege un punct $\mathbf{x}$ dintr-o submulţime (semiplan) oarecare, semnul lui $y(\mathbf{x})$ este mereu acelaşi, iar la trecerea în cealaltă submulţime semnul se schimbă. Pentru $n = 2$ vorbim de semiplan pozitiv şi semiplan negativ.

Distanţa de la origine la suprafaţa $\mathcal{S}$ se calculează ca:

$$d = \frac{|b|}{\|\mathbf{w}\|}$$

unde $\mathbf{w} = (w_1, \ldots, w_n)^t$ iar $\|\mathbf{w}\|$ este norma Euclidiană a vectorului $\mathbf{w}$, $\|\mathbf{w}\| = \sqrt{w_1^2 + \ldots w_n^2}$. Dacă $\mathbf{x_1}$ şi $\mathbf{x_2}$ sunt două puncte de pe varietatea liniară $\mathcal{S}$, atunci:

$$y(\mathbf{x_1}) = 0 \Leftrightarrow \mathbf{w}^t \cdot \mathbf{x_1} + b = 0 \quad (5.3)$$

$$y(\mathbf{x_2}) = 0 \Leftrightarrow \mathbf{w}^t \cdot \mathbf{x_2} + b = 0 \quad (5.4)$$

Scăzând (5.3) din (5.4) obţinem $\mathbf{w}^t \cdot (\mathbf{x_1} - \mathbf{x_2}) = 0$, sau, echivalent, vectorul $\mathbf{w}$ este perpendicular pe varietatea liniară $y(\mathbf{x}) = 0$, a se vedea figura 5.2.

Distanţa dintre un punct $\mathbf{x}$ şi suprafaţa $\mathcal{S}$ este:

$$z = \frac{|y(\mathbf{x})|}{\|w\|} \quad (5.5)$$

Putem renota termenul liber b din (5.1) cu w_0, putem considera că vectorul $\mathbf{x}$ mai are o componentă $x_0 = 1$; dacă notăm tot cu $\mathbf{w}$ vectorul de

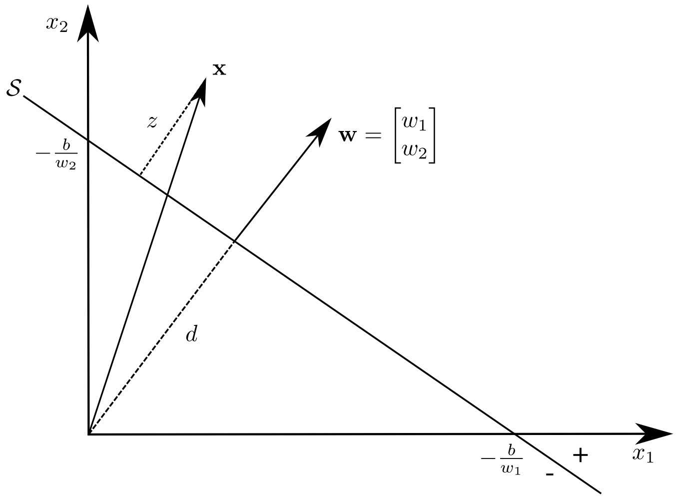


Figura 5.2: Suprafaţa de decizie $\mathcal{S}$, distanţa de la origine la $\mathcal{S}$, poziţia vectorului de ponderi $\mathbf{w}$ faţă de $\mathcal{S}$

ponderi extins $\mathbf{w} = (w_0, w_1, \ldots, w_n)^t$ şi $\mathbf{x} = (x_0, x_1, \ldots, x_n)^t$ atunci funcţia de separare y din (5.1) se rescrie ca:

$$y(\mathbf{x}) = \mathbf{w}^t \cdot \mathbf{x} \tag{5.6}$$

5.2 Perceptronul liniar

Setul de instruire este

$$S = \left\{ \left(\mathbf{x}^{(i)}, y^{(i)} \right) | \mathbf{x}^{(i)} \in \mathbb{R}^n, y^{(i)} \in \{-1, 1\}, 1 \leq i \leq m \right\} \tag{5.7}$$

$y^{(i)}$ este eticheta vectorului de intrare $\mathbf{x}^{(i)}$. Putem partiţiona vectorii de intrare $\mathbf{x}^{(i)}$ în submulţimile:

- C_1 - clasa pozitivă, acei $\mathbf{x}^{(i)} \in \mathbb{R}^n$ pentru care $y^{(i)} = +1$,
- C_2 - clasa negativă, acei $\mathbf{x}^{(i)} \in \mathbb{R}^n$ pentru care $y^{(i)} = -1$.

Setul S este astfel dat încât mulţimile C_1 şi C_2 sunt liniar separabile – aceasta fiind condiţia în care perceptronul poate învăţa să construiască o suprafaţă de separare.

Reprezentarea unui perceptron este dată în figura 5.3.

Valoarea de activare a perceptronului este $z = \mathbf{w}^t \cdot \mathbf{x} = \sum_{i=0}^{n} w_i \cdot x_i$. Valoarea de ieşire a perceptronului se calculează cu funcţia de activare "treaptă":

$$f(z) = \begin{cases} 1, & \text{dacă } z > 0 \\ -1, & \text{dacă } z < 0 \\ \text{nedefinit}, & \text{altfel} \end{cases} \tag{5.8}$$

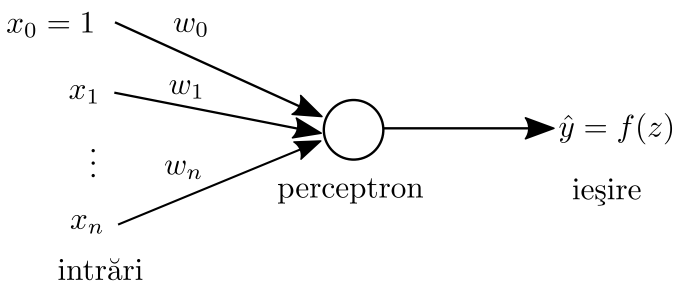


Figura 5.3: Reprezentarea unui perceptron

Valoarea produsă pentru ponderile $\mathbf{w}$ şi intrarea $\mathbf{x}$ este:

$$\hat{y} = f(\mathbf{w}^t \cdot \mathbf{x}) \tag{5.9}$$

Dacă punctul $\mathbf{x}$ este astfel încât $\mathbf{w}^t \cdot \mathbf{x} > 0$, atunci $f(\mathbf{w}^t \cdot \mathbf{x}) = 1$ şi spunem că perceptronul indică $\mathbf{x}$ ca fiind de clasă pozitivă. Dacă $\mathbf{w}^t \cdot \mathbf{x} < 0$, atunci $f(\mathbf{w}^t \cdot \mathbf{x}) = -1$ şi spunem că perceptronul indică $\mathbf{x}$ ca fiind de clasă negativă. Valoarea funcţiei f este deliberat nedefinită în 0: dacă $\mathbf{w}^t \cdot \mathbf{x} = 0$ atunci punctul $\mathbf{x}$ se află pe suprafaţa de separare şi nu se poate spune despre el că ar fi de clasă pozitivă au negativă.

5.3 Algoritmul de instruire a perceptronului

Scopul algoritmului este ca, plecând de la setul de instruire S din ecuaţia (5.7), să obţinem ponderile $\mathbf{w} = (w_0, w_1, \ldots, w_n)$ astfel încât:

$$\begin{cases} \text{pentru } \mathbf{x}^{(i)} \in C_1 \text{ să avem } \mathbf{w}^t \cdot \mathbf{x}^{(i)} > 0 \\ \text{pentru } \mathbf{x}^{(i)} \in C_2 \text{ să avem } \mathbf{w}^t \cdot \mathbf{x}^{(i)} < 0 \end{cases} \tag{5.10}$$

Se demonstrează (vezi secţiunea 5.4) că dacă C_1 şi C_2 sunt liniar separabile, atunci algoritmul de instruire poate să determine o funcţie de separare y precum în ecuaţia (5.6).

Instruirea porneşte de la ponderi aleatoare pentru valorile $w_0, w_1, \ldots, w_n$; acestea pot fi luate chiar şi 0. Notăm acest vector iniţial cu $\mathbf{w}(1)$. Printr-un proces iterativ vom obţine vectorii de ponderi $\mathbf{w}(2), \ldots, \mathbf{w}(k), \ldots$. Vectorii de ponderi se vor stabiliza la un moment dat: $\mathbf{w}(l) = \mathbf{w}(l+1) = \ldots$.

Pentru un vector de ponderi curente $\mathbf{w}(k)$, intrarea curentă $\mathbf{x}^{(i)}$ şi eticheta asociată $y^{(i)}$, estimarea produsă de perceptron este $\hat{y}^{(i)} = f\left(\mathbf{w}(k)^t \mathbf{x}^{(i)}\right)$. Dacă pentru un $\mathbf{w}(k)$ avem $\hat{y}^{(i)} = y^{(i)}$ pentru tot setul de instruire, atunci perceptronul recunoaşte corect clasele datelor din S. Altfel, este necesar să se opereze modificări pentru vectorul de ponderi curent $\mathbf{w}(k)$, detaliile fiind date în cele ce urmează.

Să considerăm ponderile de la momentul k, $\mathbf{w}(k)$. Algoritmul de instruire a perceptronului iterează peste setul de instruire. Dacă pentru o pereche $\left(\mathbf{x}^{(i)}, y^{(i)}\right) \in S$ avem că $\hat{y}^{(i)} = y^{(i)}$, atunci pentru acest caz vectorul de ponderi $\mathbf{w}(k)$ nu trebuie modificat: perceptronul clasifică corect intrarea $\mathbf{x}^{(i)}$. Dacă $\hat{y}^{(i)} \neq y^{(i)}$, atunci avem exact unul din următoarele două cazuri:

1. $\mathbf{x}^{(i)}$ e de clasă pozitivă, dar e clasificat ca negativ: $y^{(i)} = +1$ şi $\hat{y}^{(i)} = -1$

2. $\mathbf{x}^{(i)}$ e de clasă negativă, dar e clasificat ca pozitiv: $y^{(i)} = -1$ şi $\hat{y}^{(i)} = +1$

Pentru cazul 1 avem că $\mathbf{w}(k)^t \cdot \mathbf{x}^{(i)} < 0$ dar ar trebui ca produsul să fie pozitiv. Trebuie deci modificat vectorul $\mathbf{w}(k)$ astfel încât, pentru noul vector $\mathbf{w}(k+1)$ valoarea produsului scalar cu $\mathbf{x}$ să crească: $\mathbf{w}(k+1)^t \cdot \mathbf{x}^{(i)} > \mathbf{w}(k)^t \cdot \mathbf{x}^{(i)}$, în speranţa că asta va duce la $\mathbf{w}(k+1)^t \cdot \mathbf{x}^{(i)}$ şi deci la $\hat{y}^{(i)} = f(\mathbf{w}(k+1)^t \cdot \mathbf{x}^{(i)}) = +1 = y^{(i)}$. Modificarea propusă $\Delta\mathbf{w}(k)$ este:

$$\Delta\mathbf{w}(k) = \alpha \cdot \mathbf{x}^{(i)} \tag{5.11}$$

unde α este un coeficient strict pozitiv. Noul vector de ponderi va fi:

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \Delta\mathbf{w}(k) = \mathbf{w}(k) + \alpha \cdot \mathbf{x}^{(i)} \tag{5.12}$$

Verificăm că vectorul $\mathbf{w}(k+1)$ duce la creşterea valorii produsului scalar. La momentele k şi respectiv $k+1$, pentru vectorul de intrare $\mathbf{x}^{(i)}$ avem activările $z(k) = \mathbf{w}(k)^t \cdot \mathbf{x}^{(i)}$ şi $z(k+1) = \mathbf{w}(k+1)^t \cdot \mathbf{x}^{(i)}$.

Avem:

$$\begin{aligned} z(k+1) = \mathbf{w}(k+1)^t \cdot \mathbf{x} &= \left(\mathbf{w}(k) + \alpha \cdot \mathbf{x}^{(i)}\right)^t \cdot \mathbf{x} = \\ &= \mathbf{w}(k)^t \cdot \mathbf{x} + \alpha \cdot \left(\mathbf{x}^{(i)}\right)^t \cdot \mathbf{x}^{(i)} = z(k) + \left\|\mathbf{x}^{(i)}\right\|^2 \geq z(k) \end{aligned} \tag{5.13}$$

cu inegalitate strictă pentru $\left\|\mathbf{x}^{(i)}\right\| \neq 0$. Avem deci creşterea valorii de activare a perceptronului, aşa cum ne-am propus.

Pentru cazul 2, facem modificarea ponderilor astfel:

$$\Delta\mathbf{w}(k) = -\alpha \cdot \mathbf{x}^{(i)} \tag{5.14}$$

deci

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \Delta\mathbf{w}(k) = \mathbf{w}(k) - \alpha \cdot \mathbf{x}^{(i)} \tag{5.15}$$

cu aceeaşi condiţie pentru α. Se verifică, similar ca în (5.13) că $z(k+1) \leq z(k)$, cu inegalitate strictă pentru $\left\|\mathbf{x}^{(i)}\right\| \neq 0$.

Avem deci că:

$$\Delta\mathbf{w}(k) = \begin{cases} 0, & \text{dacă } y^{(i)} = \hat{y}^{(i)} \\ \alpha \cdot \mathbf{x}^{(i)}, & \text{dacă } y^{(i)} = +1 \text{ şi } \hat{y}^{(i)} = -1 \\ -\alpha \cdot \mathbf{x}^{(i)}, & \text{dacă } y^{(i)} = -1 \text{ şi } \hat{y}^{(i)} = +1 \end{cases} \tag{5.16}$$

Putem scrie modificările din ecuaţiile (5.11) şi (5.14) unificat, astfel:

$$\Delta\mathbf{w}(k) = \frac{\alpha}{2}(y^{(i)} - \hat{y}^{(i)}) \cdot \mathbf{x}^{(i)} \tag{5.17}$$

Dacă vrem să calculăm $\Delta\mathbf{w}(k)$ doar pentru acele cazuri pentru care $y^{(i)} \neq \hat{y}^{(i)}$ atunci:

$$\Delta\mathbf{w}(k) = \alpha y^{(i)}\mathbf{x}^{(i)} \tag{5.18}$$

E posibil ca o singură modificare a ponderilor să nu fie suficientă pentru ca tot setul de instruire S să fie clasificat corect. Dacă în decursul unei epoci de instruire – epocă înseamnă parcurge în întregime a setului de instruire S – apare măcar o modificare, se efectuează o nouă epocă. Procesul se încheie când ponderile se stabilizează, deci setul de instruire S e învăţat.

Algoritmul de instruire a perceptronului este:

1. Iniţializează $\mathbf{w}(1)$ cu componente aleatoare sau cu vectorul nul; $k = 1$
2. Repetă:
   (a) $corectClasificat = adevarat$
   (b) pentru $i = \overline{1, m}$
      i. calculează $\hat{y}^{(i)} = f\left(\mathbf{w}(k)^t \cdot \mathbf{x}^{(i)}\right)$
      ii. dacă $\hat{y}^{(i)} \neq y^{(i)}$ atunci:
         A. $corectClasificat = fals$
         B. $\mathbf{w}(k+1) = \mathbf{w}(k) + \alpha y^{(i)}\mathbf{x}^{(i)}$
         C. $k = k + 1$
3. Până când $corectClasificat = adevarat$
4. Vectorul $\mathbf{w}(k)$ este vectorul final de ponderi pentru perceptron, iar k este numărul total de setări ale vectorului de ponderi $\mathbf{w}$.

Deşi algoritmul modifică instantaneu ponderile pentru fiecare caz în care ieşirea furnizată de perceptron nu coincide cu eticheta cunoscută, algoritmul nu este incremental în sensul dat în secţiunea 9.1, având de regulă nevoie de mai multe epoci de instruire.

Faptul că algoritmul se opreşte după un număr finit de paşi este demonstrat în secţiunea 5.4. Notăm cu $\mathbf{w}$ vectorul de ponderi produs la terminarea algoritmului, $\mathbf{w} = \mathbf{w}(k)$.

După ce algoritmul de instruire se opreşte, perceptronul este folosit pentru a face clasificarea unui vector $\mathbf{x} \in \mathbb{R}^n$, astfel:

$$\begin{cases} \text{dacă } \mathbf{w}^t \cdot \mathbf{x} > 0, & \text{atunci } \mathbf{x} \text{ e de clasă pozitivă} \\ \text{dacă } \mathbf{w}^t \cdot \mathbf{x} < 0, & \text{atunci } \mathbf{x} \text{ e de clasă negativă} \end{cases} \tag{5.19}$$

5.4 Convergenţa perceptronului

Vom demonstra că algoritmul de instruire a perceptronului se termină în număr finit de paşi, dacă setul de instruire este compus din două mulţimi C_1, C_2 liniar separabile.

Condiţia de liniar separabilitate ne permite să spunem că există un vector $\mathbf{w}_* \in \mathbb{R}^{n+1}$ cu:

$$\begin{cases} \text{pentru } \mathbf{x}^{(i)} \in C_1 \text{ să avem } \mathbf{w}_*^t \cdot \mathbf{x}^{(i)} > 0 \\ \text{pentru } \mathbf{x}^{(i)} \in C_2 \text{ să avem } \mathbf{w}_*^t \cdot \mathbf{x}^{(i)} < 0 \end{cases} \tag{5.20}$$

(a se revedea condiţiile 5.10), sau, mai pe scurt,

$$y^{(i)} \cdot \mathbf{w}_*^t \cdot \mathbf{x}^{(i)} > 0 \text{ pentru orice } i, 1 \leq i \leq m \tag{5.21}$$

Putem presupune că vectorul $\mathbf{w}_*$ este de normă 1; în caz contrar se poate face normarea lui[1], iar situaţiile din (5.21) se menţin.

Întrucât numărul de elemente din setul de instruire este finit, putem găsi $\gamma > 0$ astfel încât:

$$y^{(i)} \cdot \mathbf{w}_*^t \cdot \mathbf{x}^{(i)} > \gamma \text{ pentru orice } i, 1 \leq i \leq m \tag{5.22}$$

de exemplu

$$\gamma = \frac{1}{2} \cdot \min_{1 \leq i \leq m} \left\{ y^{(i)} \cdot \mathbf{w}_*^t \cdot \mathbf{x}^{(i)} \right\} > 0 \tag{5.23}$$

Tot datorită faptului că mulţimea de instruire e finită, avem că există un $R > 0$ astfel încât $\|x^{(i)}\| \leq R$: toţi vectorii de intrare din setul de instruire sunt cuprinşi în interiorul unei sfere centrate în origine şi de rază R suficient de mare.

Deoarece vectorul $\mathbf{w}_*$ nu este cunoscut (ştim însă că există, pentru că C_1 şi C_2 sunt liniar separabile), cantitatea γ e şi ea necunoscută. Considerarea acestor cantităţi e totuşi utilă pentru demonstrarea convergenţei algoritmului de instruire a perceptronului. O valoare a lui R se poate găsi pe baza datelor, de exemplu

$$R = \max_{1 \leq i \leq m} \left\{ \|\mathbf{x}^{(i)}\| \right\} \tag{5.24}$$

Două presupuneri care simplifică partea de demonstraţie pentru teorema ce urmează sunt:

1. $\alpha = 1$; valoarea lui α nu are de fapt importanţă, atâta timp cât e mai mare ca zero;

2. iniţial vectorul $\mathbf{w}(1)$ are toate elementele 0.

[1] Un vector nenul împărţit la norma lui devine de normă 1

Teorema 1. *(Teorema de convergenţă a perceptronului) Algoritmul de instruire a perceptronului efectuează cel mult R^2/γ^2 modificări ale ponderilor, după care returnează un hiperplan de separare.*

Demonstraţie. Pentru $k \geq 1$, fie un $\mathbf{x}^{(i)}$ clasificat greşit de perceptron, adică $f(\mathbf{w}(k)^t \mathbf{x}^{(i)}) \neq y^{(i)}$, sau, echivalent:

$$f(\mathbf{w}(k)^t \mathbf{x}^{(i)}) \neq y^{(i)} \Leftrightarrow y^{(i)} \cdot \mathbf{w}(k)^t \mathbf{x}^{(i)} < 0 \tag{5.25}$$

Pentru această situaţie de clasificare greşită a unui vector din setul de instruire, algoritmul efectuează modificare a lui $\mathbf{w}(k)$.

Avem:

$$\begin{aligned}
\mathbf{w}(k+1)^t \cdot \mathbf{w}_* &= (\mathbf{w}(k) + y^{(i)}\mathbf{x}^{(i)})^t \cdot \mathbf{w}_* && (5.26)\\
&= \mathbf{w}(k)^t \cdot \mathbf{w}_* + y^{(i)}(\mathbf{x}^{(i)})^t \cdot \mathbf{w}_* && (5.27)\\
&> \mathbf{w}(k)^t \cdot \mathbf{w}_* + \gamma && (5.28)
\end{aligned}$$

Prin inducţie matematică şi ţinând cont că $\mathbf{w}(1) = \mathbf{0}$, se arată că:

$$\mathbf{w}(k+1)^t \cdot \mathbf{w}_* > k\gamma \tag{5.29}$$

Folosim inegalitatea Cauchy-Schwartz ($|\mathbf{a}^t \cdot \mathbf{b}| \leq \|\mathbf{a}\|\|\mathbf{b}\|$), $a \leq |a| \forall a \in \mathbb{R}$ şi obţinem:

$$\mathbf{w}(k+1)^t \cdot \mathbf{w}_* \leq |\mathbf{w}(k+1)^t \cdot \mathbf{w}_*| \leq \|\mathbf{w}(k+1)\| \cdot \|\mathbf{w}_*\| = \|\mathbf{w}(k+1)\| \tag{5.30}$$

şi din ultimele două inegalităţi avem:

$$\|\mathbf{w}(k+1)\| > k\gamma \tag{5.31}$$

Pe de altă parte, avem că:

$$\begin{aligned}
\|\mathbf{w}(k+1)\|^2 &= \|\mathbf{w}(k) + y^{(i)}\mathbf{x}^{(i)}\| && (5.32)\\
&= \|\mathbf{w}(k)\|^2 + \|y^{(i)}\mathbf{x}^{(i)}\|^2 + 2\mathbf{w}(k)^t \cdot \mathbf{x}^{(i)} \cdot y^{(i)} && (5.33)\\
&= \|\mathbf{w}(k)\|^2 + |y^{(i)}|^2 \cdot \|\mathbf{x}^{(i)}\|^2 + 2\mathbf{w}(k)^t \cdot \mathbf{x}^{(i)} \cdot y^{(i)} && (5.34)\\
&\leq \|\mathbf{w}(k)\|^2 + \|\mathbf{x}^{(i)}\|^2 && (5.35)\\
&\leq \|\mathbf{w}(k)\|^2 + R^2 && (5.36)
\end{aligned}$$

Ţinând cont că $\mathbf{w}(1) = \mathbf{0}$, prin inducţie matematică se arată că:

$$\|\mathbf{w}(k+1)\|^2 \leq kR^2 \tag{5.37}$$

unde trecerea de la (5.34) la (5.35) se face pe baza inegalităţii (5.25).

Din inecuaţiile (5.31) şi (5.37) rezultă că:

$$k^2\gamma^2 < \|\mathbf{w}(k+1)\|^2 \leq kR^2 \tag{5.38}$$

de unde $k < \frac{R^2}{\gamma^2}$.

Am obţinut deci că indicele k pentru care se fac modificare de ponderi nu poate fi oricât de mare, adică algoritmul se termină în timp finit. Finalizarea lui înseamnă totodată obţinerea unui set de ponderi pentru forma de separare liniară care să clasifice corect cazurile din setul de instruire. □

Comentariu: Deoarece valoarea lui γ nu e cunoscută, rezultatul de mai sus nu ne spune practic care e numărul efectiv de paşi necesari. Totuşi, există o dovadă a faptului că algoritmul nu rulează la infinit.

5.5 Algoritmul lui Gallant

Algoritmul lui Pocket – sau algoritmul "buzunarului" – tratează cazul în care setul de instruire nu este liniar separabil. Ideea algoritmului este de a menţine vectorul $\mathbf{w}$ de ponderi care face cele mai puţine erori de clasificare pentru date succesive. La finalul unei epoci se contorizează câte cazuri din setul de instruire sunt corect clasificate de vectorul curent de ponderi. Dacă numărul de clasificări corecte este mai mare decât pentru vectorul menţinut până până acum într-un "buzunar" (la început: vectorul $\mathbf{w}(1)$ cu număr de clasificări corecte cunoscute 0), atunci se actualizează conţinutul "buzunarului": vectorul curent de ponderi şi numărul de clasificări corecte. Procesul se repetă de un număr de ori specificat. La final se returnează vectorul de ponderi din "buzunar".

5.6 Comentarii

Spre deosebire de regresia logistică, perceptronul liniar nu produce o valoare care să exprime în ce măsură modelul consideră că un vector de intrare aparţine clasei pozitive, $P(C_1|\mathbf{x})$. Totuşi, vine cu demonstraţie matematică pentru convergenţă, dacă un separator liniar desparte clasa pozitivă de cea negativă.

La momentul apariţiei, perceptronul liniar a fost privit ca un motiv clar pentru care problemele cele mai complexe sunt rezolvabile prin perceptroni. Cartea *Perceptrons*[2] a lui Minsky si Papert, din 1969, arată însă că perceptronul nu poate rezolva probleme care sunt neseparabile liniar, de exemplu problema XOR. Conjectura lor că utilizarea de mai mulţi perceptroni nu poate să ducă la rezolvarea de probleme neseparabile liniar a devenit extrem de populară, motiv pentru care cercetările în domeniul reţelelor neurale artificiale au fost descurajate. Revenirea s–a produs în 1986, când

[2] Marvin Minsky şi Seymour Papert, *Perceptrons: an introduction to computational geometry*, , MIT Press, 1969.

Rumelhart, Hinton şi Williams[3] au propus o procedură de învăţare pentru reţele cu mai multe straturi de neuroni neliniari care permitea abordarea claselor neseparabile liniar.

Setul de instruire pentru problema XOR este următorul:

$$\left\{((0,0)^t,0),((1,1)^t,0),((1,0)^t,1),((0,1)^t,1)\right\}$$

unde fiecare din cele 4 tuple contine o pereche de valori de intrare din $\{0,1\}^2$ împreună cu eticheta de clasă asociată, 0 sau 1. Se poate demonstra algebric că nu există un vector de 3 ponderi $(w_0, w_1, w_2)^t \in \mathbb{R}^3$ pentru care:

$$sgn(w_0 \cdot 1 + w_1 \cdot 0 + w_2 \cdot 0) = sgn(w_0 \cdot 1 + w_1 \cdot 1 + w_2 \cdot 1) = 1 \tag{5.39}$$

iar

$$sgn(w_0 \cdot 1 + w_1 \cdot 0 + w_2 \cdot 1) = sgn(w_0 \cdot 1 + w_1 \cdot 1 + w_2 \cdot 0) = -1 \tag{5.40}$$

Pentru problema n–dimensională, clasa de ieşire pentru vectorul binar $\mathbf{x} \in \{0,1\}^n$ este 1 dacă numărul de componente 1 este impar, altfel 0. Se poate arăta că şi pentru cazul n dimensional problema nu e rezolvabilă printr–un separator liniar.

[3]David E. Rumelhart, Geoffrey E. Hinton, Ronald J. Williams, *Learning representations by back-propagating errors*, Nature, volume 323, issue 6088, pp. 533-536, 1986.

Capitolul 6

Perceptronii multistrat

Reţelele neurale multistrat — sau perceptronii multistrat, multilayer perceptrons (MLPs) — sunt folosite pentru probleme de regresie, de clasificare şi de estimare de probabilităţi condiţionate. Instruirea este supervizată. Sunt cea mai populară variantă de reţele neurale artificiale şi fac parte din familia mai mare a reţelelor cu propagare înainte.

6.1 Motivaţie pentru reţele neurale multistrat

Conform celor din cursul precedent, un perceptron liniar este capabil să găsească un hiperplan de separare pentru două mulţimi, dacă – şi numai dacă – ele sunt liniar separabile. Există însă exemple de probleme simple care nu sunt liniar separabile — şi deci nerezolvabile de către perceptronul liniar — dar care pot fi totuşi separate. În plus, dorim să rezolvăm şi altfel de probleme decât de clasificare binară: regresie (estimare de valoare de ieşire de tip continuu), estimare de probabilitate condiţionată, clasificare pentru mai mult de două clase. Cursul de faţă conţine modele bazate pe neuroni neliniari în care se pot rezolva toate aceste tipuri de probleme.

Figura 6.1 conţine cea mai celebră problemă care nu poate fi rezolvată prin clasificator liniar: problema XOR. Se consideră setul

$$\mathcal{S} = \{((0,0),0),((0,1),1),((1,0),1),((1,1),0)\}$$

unde fiecare din cele patru elemente este compus din pereche de intrare $(x, y) \in \{0,1\}^2$ şi dintr–o etichetă de clasă binară. Se poate demonstra algebric că într–adevăr nu se pot separa cele două clase printr–o dreaptă (punctele de clasă 0 să fie de o parte a dreptei, cele de clasa 1 de cealaltă parte). Un alt exemplu este dat în figura 6.2 [1].

Intuim că un singur neuron e prea puţin pentru probleme complexe de separare. Pe de altă parte, concatenarea mai multor neuroni cu funcţie de activare liniară este echivalentă cu produsul dintre înmulţirea unei secvenţe de matrice cu vectorul de intrare. Datorită faptului că înmulţirea de matrice

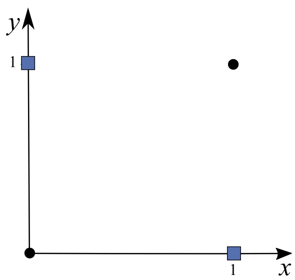


Figura 6.1: Problema XOR. Clasele sunt marcate cu forme şi culori diferite. Se poate demonstra că nu se poate trasa o dreaptă în plan care să aibă de o parte a ei doar puncte din clasa "0" şi de cealaltă parte doar puncte de clasă "1".

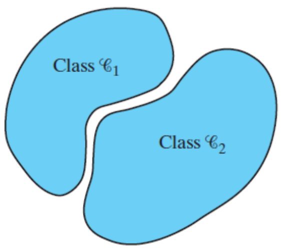


Figura 6.2: Două clase de puncte ce nu sunt separabile liniar [1].

produce tot o matrice, operaţia este echivalentă cu înmulţirea unei matrice cu vectorul de intrare. Obţinem de aici că succesiunea de neuroni cu funcţie de activare liniară este echivalentă cu un singur neuron cu funcţie de activare liniară.

Vom folosi deci mai mulţi neuroni, iar funcţiile lor de activare vor fi neliniare.

6.2 Notaţii folosite

Tabelul 6.1 conţine notaţiile principale care se folosesc în acest curs.

6.3 Setul de instruire

Reţelele din acest capitol sunt pentru instruire de tip supervizat. Setul de instruire este de forma:

$$\mathcal{S} = \left\{ \left(\mathbf{x}^{(1)}, \mathbf{d}^{(1)}\right), \left(\mathbf{x}^{(2)}, \mathbf{d}^{(2)}\right), \dots, \left(\mathbf{x}^{(p)}, \mathbf{d}^{(p)}\right) \right\} \tag{6.1}$$

unde $\mathbf{x}^{(i)} \in \mathbb{R}^n$ iar $\mathbf{d}^{(i)}$ este, după caz:

- pentru o problemă de regresie: vector oarecare din $\mathbb{R}^m$;

Noţiune sau notaţie	Explicaţie
p	numărul de perechi în setul de instruire
$\mathbf{x}^{(i)}$	vector de intrare din setul de instruire, $\mathbf{x}^{(i)} = \left(\mathbf{x}_1^{(i)}, \dots, \mathbf{x}_n^{(i)}\right)^t$, $1 \le i \le p$
$\mathbf{d}^{(i)}$	ieşirea asociată intrării $\mathbf{x}^{(i)}$, din setul de instruire, $\mathbf{d}^{(i)} = \left(\mathbf{d}_1^{(i)}, \dots, \mathbf{d}_m^{(i)}\right)^t$, $1 \le i \le p$
L	numărul de straturi din reţeaua neurală, inclusiv straturile de intrare şi de ieşire
nod	neuron – dacă apare în stratul $1, 2, \dots, L-1$ – sau nod de intrare – dacă apare în primul strat (de indice 0)
n_l	numărul de noduri din stratul l, $0 \le l \le L-1$;
$z_i^{[l]}$	valoarea de activare a neuronului i din stratul l, pentru $1 \le l \le L-1$, $0 \le i \le n_l$
$\mathbf{z}^{[l]}$	vectorul conţinând valorile de activare ale neuronilor din stratul l, $\mathbf{z}^{[l]} = \left(z_1^{[l]}, \dots, z_{n_l}^{[l]}\right)^t$, $1 \le l \le L-1$
$a_i^{[l]}$	valoarea de ieşire a celui de al i–lea nod din stratul l, pentru $0 \le l \le L-1$, $1 \le i \le n_l$
$\mathbf{a}^{[l]}$	vectorul cu valorile de ieşire ale nodurilor din stratul l, pentru $0 \le l \le L-1$, $\mathbf{a}^{[l]} = \left(\mathbf{a}_1^{[l]}, \dots, \mathbf{a}_{n_l}^{[l]}\right)^t$
$w_{ij}^{[l]}$	ponderea legăturii între neuronul i de pe stratul l şi nodul j de pe stratul $l-1$, $1 \le l \le L-1$, $1 \le i \le n_l$, $1 \le j \le n_{l-1}$
$\mathbf{W}^{[l]}$	matricea de ponderi dintre stratul $l-1$ şi stratul l, $0 \le l \le L-1$, $\mathbf{W}_{ij}^{[l]} = w_{ij}^{[l]}$, $1 \le i \le n_l$, $1 \le j \le n_{l-1}$
$\mathbf{W}_i^{[l]}$	linia i a matricei $\mathbf{W}^{[l]}$, $1 \le l \le L-1$, $1 \le i \le n_l$
$b_i^{[l]}$	ponderea de bias pentru neuronul i din stratul l, $1 \le l \le L-1$, $1 \le i \le n_l$
$\mathbf{b}^{[l]}$	vectorul ponderilor de bias către stratul l, $\mathbf{b}^{[l]} = \left(b_1^{[l]}, \dots, b_{n_l}^{[l]}\right)^t$, $1 \le l \le L-1$
$f^{[l]}$	funcţie de activare a neuronilor din stratul l, $1 \le l \le L-1$
$\mathbf{W}$	secvenţa de matrice de ponderi $(\mathbf{W}^1, \mathbf{W}^2, \dots, \mathbf{W}^{L-1})$
$\mathbf{b}$	secvenţa de vectori de ponderi de bias $(\mathbf{b}^1, \mathbf{b}^2, \dots, \mathbf{b}^{L-1})$
$J(\mathbf{W}, \mathbf{b})$	eroare empirică pentru set de instruire sau minibatch
$J\left(\mathbf{W}, \mathbf{b}; \mathbf{x}^{(i)}, \mathbf{d}^{(i)}\right)$	eroarea pentru perechea de vectori de instruire $\left(\mathbf{x}^{(i)}, \mathbf{d}^{(i)}\right)$, $1 \le i \le p$
$\mathbf{o}^{(i)}$	vector coloană de ieşire corespunzător intrării $\mathbf{x}^{(i)}$, calculat de reţea
$\boldsymbol{\delta}^l$	semnalul de eroare pentru stratul l, $1 \le l \le L-1$
$\odot$	produs Hadamard

Tabela 6.1: Notaţii folosite

- pentru o problemă de clasificare sau estimare de probabilităţi pentru m clase: vectori de forma $(1, 0, \ldots, 0)$, $(0, 1, 0, \ldots, 0)$, $\ldots$, $(0, 0, \ldots, 0, 1)$ cu m valori binare, din care doar cea de pe poziţia aferentă clasei curente este unu iar restul sunt zero[1].

6.4 Reţeaua neurală multistrat

6.4.1 Arhitectură

Există mai multe modalităţi de dispunere a neuronilor; noi vom folosi o arhitectură de tip multistrat, feedforward, numită perceptron multistrat – chiar dacă neuronii folosiţi nu sunt perceptroni, ci neuroni cu funcţie de activare neliniară. O reţea multistrat se compune din minim trei straturi:

- strat de intrare ce preia valorile de intrare; nu are rol computaţional, nu este format din neuroni[2];
- măcar un strat ascuns, compus din neuroni;
- strat de ieşire, de asemenea compus din neuroni, produce valori estimate care sunt apoi comparate cu ieşirile dorite.

Numerotarea straturilor începe de la 0. Neuronii ascunşi produc trăsături noi pe baza vectorilor de intrare, trăsături care sunt mai apoi necesare reţelei neurale pentru producerea unei estimări. Este posibil ca o reţea să aibă mai mult de un neuron în stratul de ieşire, aşa cum se vede în figura 6.4.

Se consideră că instruirea e mai eficientă dacă pe lângă valorile de intrare şi pe lângă valorile calculate de un strat de neuroni se mai furnizează o valoare constantă, de regulă $+1$, înmulţită cu o pondere de *bias*[3]. Ponderile dintre straturi precum şi aceste ponderi de *bias* sunt instruibile, adică se vor modifica prin procesul de învăţare[4].

O reprezentare de reţea neurală cu trei straturi şi o ieşire este dată în figura 6.3; o reţea cu 4 straturi şi două ieşiri este reprezentată în figura 6.4. Nu există o relaţie anume între numărul de straturi ascunse şi numărul de neuroni de intrare şi ieşire.

Vom considera că avem $L \geq 3$ straturi şi în fiecare strat ascuns l $(1 \leq l \leq L - 1)$ un număr de n_l neuroni. Stratul de intrare are $n_0 = n$ noduri,

[1] Aşa–numita codificare *one–hot* sau *1-din-m*.

[2] Motiv pentru care unii autori nu îl consideră un strat propriu–zis; frecvent se foloseşte exprimarea că o reţea are "k" straturi ascunse, cele de intrare şi ieşire fiind oricum existente. În cele ce urmează considerăm intrarea ca formând un strat.

[3] Unii autori consideră valoarea constantă -1; nu este esenţial, deoarece ponderile sunt coeficienţi ce pot avea orice semn.

[4] O discuţie asupra necesităţii considerării bias–ului este la ftp://ftp.sas.com/pub/neural/FAQ2.html#A_bias

numărul de neuroni din stratul de ieşire este $n_{L-1} = m$ dat de: numărul de clase pentru care se face recunoaşterea (la problemă de clasificare sau estimare de probabilitate condiţionată) respectiv numărul de ieşiri care se doresc a fi aproximate (la regresie).

Pentru oricare dintre figuri:

- valorile x_1, x_2, x_3 sunt componentele vectorului de intrare $\mathbf{x} = (x_1, x_2, x_3)^t$; se mai consideră încă o intrare cu valoarea constantă $+1$, aferentă *bias*–ului;
- valoarea $a_i^{[l]}$ este ieşirea nodului i din stratul l, $0 \leq l \leq L-1$, $1 \leq i \leq n_l$; se remarcă în figuri prezenţa valorilor constante $+1$ în toate straturile, exceptând cel de ieşire. Pentru stratul de intrare, $a_i^{[0]} = x_i$;
- valoarea $h_{\mathbf{W},\mathbf{b}}(\mathbf{x})$ este ieşirea calculată de către reţea pentru vectorul curent de intrare $\mathbf{x}$; $h_{\mathbf{W},\mathbf{b}}(\mathbf{x}) \in \mathbb{R}$ pentru figura 6.3 şi $h_{\mathbf{W},\mathbf{b}}(\mathbf{x}) \in \mathbb{R}^2$ pentru figura 6.4.

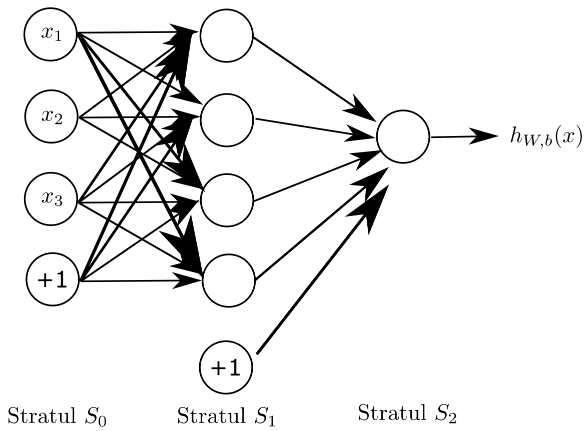


Figura 6.3: Reţea MLP cu 3 straturi

Perechea $(\mathbf{W}, \mathbf{b})$ formează mulţimea ponderilor şi a valorilor de bias din în reţea. Folosim următoarele notaţii:

- ponderile dintre stratul de intrare şi stratul ascuns sunt conţinute în matricea $\mathbf{W}^{[1]}$: $w_{ij}^{[1]}$ este ponderea legăturii dintre neuronul i al stratului al doilea (de indice 1) şi nodul j din stratul de intrare; se remarcă ordinea indicilor inferiori, utilă mai departe pentru operaţiile de algebră liniară ce vor fi folosite;
- în general, notăm cu $w_{ij}^{[l]}$ ponderea legăturii dintre al i–lea neuron din stratul de indice l şi al j–lea nod (neuron sau nod de intrare) din stratul precedent $l-1$ $(1 \leq l \leq L-1)$;

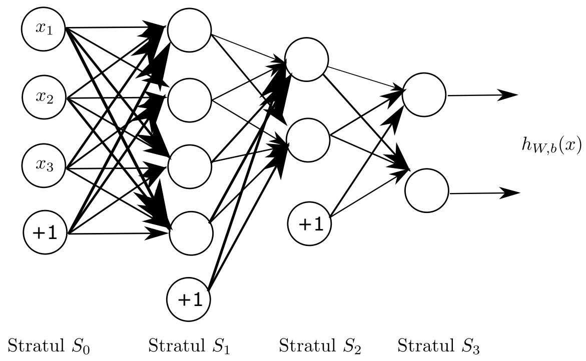


Figura 6.4: Reţea MLP cu 4 straturi

- valoarea ponderii dintre intrarea constantă +1 din stratul de intrare şi neuronul i din primul strat ascuns este stocată de $b_i^{[1]}$, $1 \leq i \leq n_2$;
- în general, coeficientul de bias provenind din stratul $l-1$ $(1 \leq l \leq L-1)$ şi care afectează neuronul i din stratul l este notat cu $b_i^{[l]}$, $1 \leq i \leq n_l$.

În ce priveşte numărul de ponderi instruibile – atât cele din matricele $\mathbf{W}^{[l]}$ cât şi ponderile de bias – avem, pentru $1 \leq l \leq L-1$:

- matricea $\mathbf{W}^{[l]}$ de ponderi dintre stratul $l-1$ şi stratul l are n_l linii şi n_{l-1} coloane;
- vectorul coloană de coeficienţi bias $\mathbf{b}^{[l]}$ conţine n_l valori, având forma $\mathbf{b}^{[l]} = \left(b_1^{[l]}, b_2^{[l]}, \ldots, b_{n_l}^{[l]}\right)^t$.

6.4.2 Funcţii de activare

Fiecare neuron agregă valorile din nodurile din stratul anterior – incluzând şi termenul constant +1 multiplicat cu coeficientul de bias. Neuronul de indice i din stratul $l \geq 1$ are valoarea de activare calculată ca:

$$\begin{aligned} z_i^{[l]} &= w_{i1}^{[l]} \cdot a_1^{[l-1]} + w_{i2}^{[l]} \cdot a_2^{[l-1]} + \cdots + w_{i,n_{l-1}}^{[l]} \cdot a_{n_{l-1}}^{[l-1]} + b_i^{[l]} && (6.2) \\ &= \mathbf{W}_i^{[l]} \cdot \mathbf{a}^{[l-1]} + b_i^{[l]},\ 1 \leq i \leq n_l && (6.3) \end{aligned}$$

unde: $\mathbf{W}_i^{[l]}$ este linia i a matricei $\mathbf{W}^{[l]}$, $a_i^{[l-1]}$ este, după caz: valoarea de ieşire a neuronului i din stratul $l-1$ (dacă $l \geq 1$) sau valoarea x_i din vectorul de intrare curent (dacă $l = 0$). Notând cu $\mathbf{z}^{[l]}$ vectorul coloană $\left(z_1^{[l]}, z_2^{[l]}, \ldots, z_{n_l}^{[l]}\right)^t$, putem scrie matricial:

$$\mathbf{z}^{[l]} = \mathbf{W}^{[l]} \cdot \mathbf{a}^{[l-1]} + \mathbf{b}^{[l]},\ 1 \leq l \leq L-1 \tag{6.4}$$

Pe baza valorii de activare $z_i^{[l]}$ a neuronului i din stratul l se calculează ieşirea sa folosind funcţia de activare $f^{[l]}$:

$$a_i^{[l]} = f^{[l]}\left(z_i^{[l]}\right) \tag{6.5}$$

pentru $1 \leq l \leq L-1, 1 \leq i \leq n_l$. Dacă folosim notaţia $f^{[l]}\left((z_1, z_2, \ldots, z_k)^t\right) \stackrel{\text{def}}{=} \left(f^{[l]}(z_1), f^{[l]}(z_2), \ldots, f^{[l]}(z_k)\right)^t$ – adică se aplică funcţia $f^{[l]}$ pe fiecare valoare din vectorul argument, de exemplu prin vectorizare – atunci putem scrie mai compact ecuaţia (6.5) sub forma:

$$\mathbf{a}^{[l]} = f^{[l]}\left(\mathbf{z}^{[l]}\right) \tag{6.6}$$

Funcţia $f(\cdot)$ este necesară în pasul de propagare înainte; pentru pasul de propagare înapoi a erorii este folosită derivata ei. Funcţia de activare poate avea formele[5,6]:

1. **funcţia logistică sigmoidă:**

$$f = \sigma : \mathbb{R} \to (0, 1), f(z) = \sigma(z) = \frac{1}{1 + \exp(-z)} \tag{6.7}$$

   Derivata acestei funcţii este:

$$f'(z) = \sigma'(z) = \sigma(z)(1 - \sigma(z)) = f(z) \cdot (1 - f(z)) \tag{6.8}$$

2. **funcţia tangentă hiperbolică**:

$$f = \tanh : \mathbb{R} \to (-1, 1), f(z) = \tanh(z) = \frac{\exp(z) - \exp(-z)}{\exp(z) + \exp(-z)} \tag{6.9}$$

   a cărei derivată este:

$$f'(z) = \tanh'(z) = 1 - \tanh^2(z) = 1 - f^2(z) \tag{6.10}$$

   Se arată uşor că între cele două funcţii de activare tanh şi σ există relaţia:

$$\tanh(z) = 2 \cdot \sigma(2z) - 1 \tag{6.11}$$

   În practică funcţia tanh dă rezultate mai bune decât sigmoida logistică. O explicaţie teoretică se găseşte în [8]; demonstraţii empirice sunt în [9].

[5] Alte funcţii de activare mai pot fi considerate, de exemplu combinaţii liniare de polinoame Hermite.

[6] Pentru simplificarea notaţiilor, în lista de mai jos vom omite indicele superior referind numărul stratului.

3. **funcţia liniară:**
$$f(z) = a \cdot z + b \tag{6.12}$$
cu derivata $f'(z) = a$; frecvent se iau $a = 1$, $b = 0$. Este utilizată dacă se doreşte ca la ieşire valorile să fie în afara intervalelor $(0, 1)$ şi $(-1, 1)$, cum se întâmplă la funcţiile de activare de mai sus.

4. **funcţia softmax:**
$$softmax(\mathbf{z}; c) = \frac{\exp(z_c)}{\sum\limits_{i=1}^{m} \exp(z_i)} \tag{6.13}$$
unde c este indicele neuronului, iar m este numărul total de neuroni din stratul său. Funcţia softmax a mai fost folosită la regresia logistică pentru cazul a mai mult de două clase; $\mathbf{z}$ este vector cu valori de activare, $\mathbf{z} = (z_1, \dots, z_m)^t$.

   Funcţia softmax este utilă pentru a transforma un vector de valori oarecare în distribuţie de probabilitate: se arată uşor că $softmax(\mathbf{z}; c) \in (0, 1)$ $\forall c$ şi $\sum_{c=1}^{m} softmax(\mathbf{z}; c) = 1$. De regulă, $softmax$ se foloseşte pentru stratul de ieşire şi valorile furnizate se interpretează convenabil drept probabilitatea ca intrarea curentă să fie de clasă c, $1 \leq c \leq m$; clasificarea se face găsind acel indice $1 \leq c \leq m$ pentru care $softmax(\mathbf{z}; c)$ este maxim. Se utilizează în stratul de ieşire a reţelei neurale de clasificare sau estimare de probabilitate.

   Derivatele parţiale ale funcţiei softmax sunt:
$$\frac{\partial softmax(\mathbf{z}; i)}{\partial z_j} = \begin{cases} softmax(\mathbf{z}; i) \cdot (1 - softmax(\mathbf{z}; i)) & \text{dacă } i = j \\ -softmax(\mathbf{z}; i) \cdot softmax(\mathbf{z}; j) & \text{dacă } i \neq j \end{cases} \tag{6.14}$$

5. **funcţia Rectified Linear Unit (ReLU):**
$$f(z) = \max(0, z) = \begin{cases} 0 & \text{dacă } z \leq 0 \\ z & \text{dacă } z > 0 \end{cases} \tag{6.15}$$
Derivatele pe subintervale sunt uşor de calculat şi cu execuţie rapidă. În plus, spre deosebire de sigmoida logistică şi de tangenta hiperbolică, ele nu saturează.

   Reprezentarea grafică e dată în figura 6.5.

   Chiar dacă funcţia este liniară pe porţiuni, ea este neliniară în ansamblu. Faptul că doar într–un punct nu e derivabilă nu deranjează în practică.

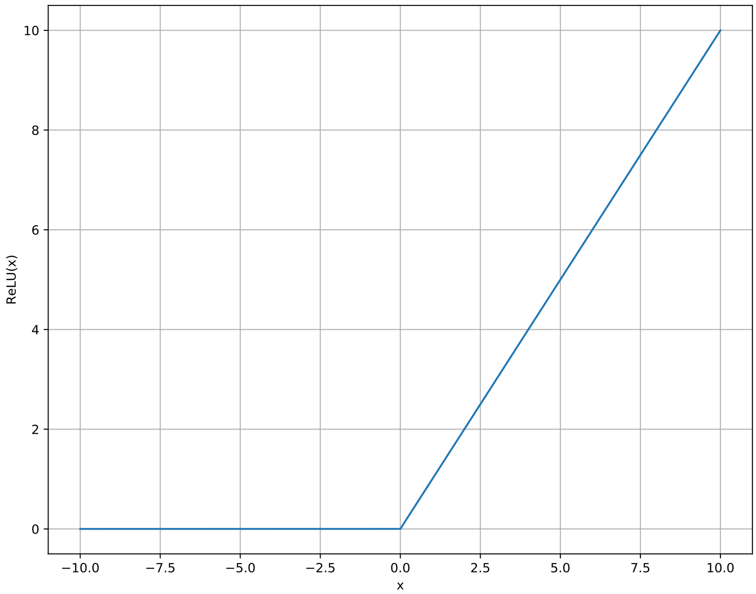


Figura 6.5: Graficul funcţiei de activare ReLU

6. **funcţia Parametric ReLU (PReLU)**:
$$f(z) = \begin{cases} \alpha z & \text{dacă } z \leq 0 \\ z & \text{dacă } z > 0 \end{cases} \tag{6.16}$$
unde $\alpha > 0$ [10], reprezentând o uşoară generalizare a funcţiei ReLU. Graficul funcţiei pentru $\alpha = 0.1$ este dat în figura 6.6.

   Pentru $\alpha = 0.01$ se obţine un caz particular vehiculat în literatură, Leaky ReLU.

7. **funcţia Exponential Linear Units (ELU)** are forma:
$$f(z) = \begin{cases} a(e^z - 1)z & \text{dacă } z \leq 0 \\ z & \text{dacă } z > 0 \end{cases} \tag{6.17}$$

Este admis ca funcţia de activare să difere de la strat la strat sau de la neuron la neuron.

6.5 Pasul de propagare înainte

Odată ce arhitectura reţelei e fixată – numărul de straturi ascunse şi numărul de neuroni în fiecare strat precum şi funcţiile de activare – se poate

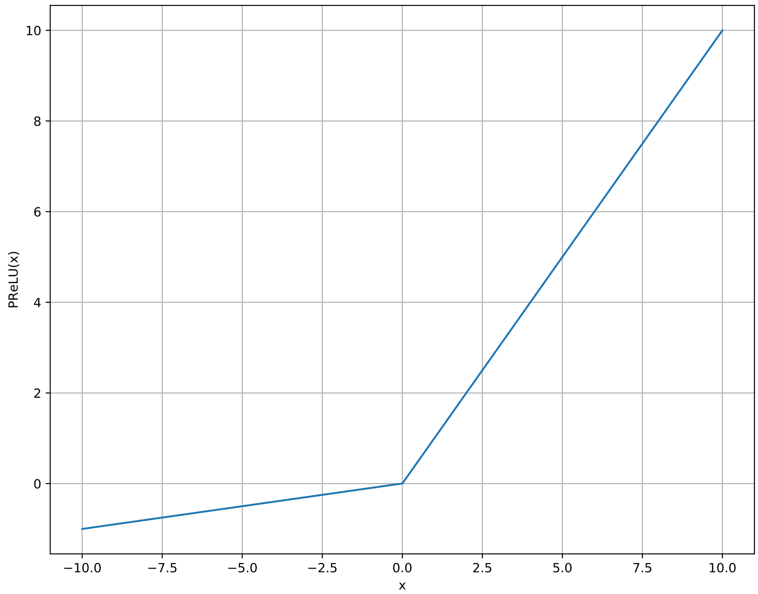


Figura 6.6: Graficul funcţiei de activare PReLU pentru $\alpha = 0.1$

trece la instruirea şi apoi utilizarea ei. Pasul de propagare înainte preia un vector de intrare $\mathbf{x} = (x_1, \ldots, x_n)^t$ şi produce modificări în starea neuronilor reţelei pornind de la intrare, acţionând succesiv asupra straturilor $1, \ldots, L-1$. Aceasta dă şi numele familiei din care face parte reţeaua: "cu propagare înainte", sau "feedforward". Ieşirile din ultimul strat sunt folosite pentru predicţie – regresie, estimare de probabilitate condiţionată sau clasificare.

După cum s–a mai afirmat, stratul de intrare nu are rol computaţional; valoare sa de ieşire este chiar vectorul de intrare – considerat ca vector coloană – $\mathbf{x}$ furnizat reţelei:

$$\mathbf{a}^{[0]} = \mathbf{x} \tag{6.18}$$

Dacă se cunosc valorile de ieşire ale nodurilor din stratul $l-1$ se pot calcula valorile de activare ale neuronilor din stratul l şi apoi valorile lor de ieşire, astfel:

$$\mathbf{z}^{[l]} = \mathbf{W}^{[l]} \cdot \mathbf{a}^{[l-1]} + \mathbf{b}^{[l]} \tag{6.19}$$

$$\mathbf{a}^{[l]} = f^{[l]}(\mathbf{z}^{[l]}) \tag{6.20}$$

pentru $l = 1, \ldots L-1$, cu $f^{[l]}(\cdot)$ funcţie de activare ce se aplică pe fiecare componentă a vectorului. Vom nota cu $\mathbf{o}$ vectorul de m valori de ieşire produs de către reţea:

$$\mathbf{o} = \mathbf{a}^{[L-1]} \tag{6.21}$$

Pentru cazul în care se lucrează pe un set de date format din perechi vectori intrare–ieşire, vectorii coloană de date de intrare se pot concatena pe orizontală într–o matrice $\mathbf{X}$. De exemplu, datele din setul de instruire $\mathcal{S} = \left\{\left(\mathbf{x}^{(1)}, \mathbf{d}^{(1)}\right), \left(\mathbf{x}^{(2)}, \mathbf{d}^{(2)}\right), \dots, \left(\mathbf{x}^{(p)}, \mathbf{d}^{(p)}\right)\right\}$, matricea $\mathbf{X}$ se formează ca:

$$\mathbf{X} = \begin{pmatrix} | & | & \dots & | \\ \mathbf{x}^{(1)} & \mathbf{x}^{(2)} & \dots & \mathbf{x}^{(p)} \\ | & | & \dots & | \end{pmatrix} \tag{6.22}$$

Propagarea înainte se efectuează cu paşii:

$$\mathbf{Z}^{[1]} = \mathbf{W}^{[1]}\mathbf{X} + \mathbf{b}^{[1]} \tag{6.23}$$
$$\mathbf{A}^{[1]} = f^{[1]}\left(\mathbf{Z}^{[1]}\right) \tag{6.24}$$
$$\mathbf{Z}^{[2]} = \mathbf{W}^{[2]}\mathbf{A}^{[1]} + \mathbf{b}^{[2]} \tag{6.25}$$
$$\mathbf{A}^{[2]} = f^{[2]}\left(\mathbf{Z}^{[2]}\right) \tag{6.26}$$
$$\dots \tag{6.27}$$
$$\mathbf{Z}^{[L-1]} = \mathbf{W}^{[L-1]}\mathbf{A}^{[L-2]} + \mathbf{b}^{[L-1]} \tag{6.28}$$
$$\mathbf{A}^{[L-1]} = f^{[L-1]}\left(\mathbf{Z}^{[L-1]}\right) \tag{6.29}$$

unde $\mathbf{Z}^{[l]}$ şi $\mathbf{A}^{[l]}$ sunt matrice cu n_l linii şi p coloane. Pentru adunările din (6.23, 6.25, 6.28) se consideră că se aplică mecanismul de "broadcasting": vectorii $\mathbf{b}^{[l]}$ produc prin copiere o matrice cu acelaşi număr de coloane ca şi matricele $\mathbf{X}$, respectiv $\mathbf{Z}^{[l-1]}$.

Se recomandă ca operaţiile date mai sus să fie implementate folosind biblioteci optimizate de algebră liniară, ce permit înmulţirea eficientă de matrice şi calcul vectorizat pe CPU sau GPU — Octave, Matlab, Numpy, PyTorch, Tensorflow etc.

6.6 Funcţii de cost

Fiecare pereche $\left(\mathbf{x}^{(i)}, \mathbf{d}^{(i)}\right) \in \mathcal{S}$ $(1 \le i \le p)$ va produce valoare pentru funcţia de eroare astfel: se furnizează vectorul $\mathbf{x}^{(i)}$ ca intrare în reţea şi se calculează un vector de ieşire $\mathbf{o}^{(i)}$, reprezentând estimarea produsă de reţea pentru intrarea furnizată; se foloseşte o funcţie de cost, sau de eroare, $J\left(\mathbf{W}, \mathbf{b}; \mathbf{x}^{(i)}, \mathbf{d}^{(i)}\right)$ care se doreşte a fi cu atât mai mică cu cât vectorul $\mathbf{o}^{(i)}$ e mai apropiat de $\mathbf{d}^{(i)}$, şi cu atât mai mare cu cât cei doi vectori sunt mai depărtaţi. În plus, se mai consideră un factor de regularizare care împiedică ponderile să devină prea mari în valoare absolută, caz asociat de regulă cu un comportament instabil al reţelei: variaţii mici ale intrării duc la salturi mari în straturile ascunse şi la ieşire.

Forma generală a funcţiei de eroare este:

$$J(\mathbf{W},\mathbf{b}) = \underbrace{\left[\frac{1}{p}\sum_{i=1}^{p} \overbrace{J\left(\mathbf{W},\mathbf{b};\mathbf{x}^{(i)},\mathbf{d}^{(i)}\right)}^{\text{eroare empirică pentru}\left(\mathbf{x}^{(i)},\mathbf{d}^{(i)}\right)}\right]}_{\text{Eroarea empirică pe tot setul de antrenare}} + \underbrace{\frac{\lambda}{2}\sum_{l=1}^{L-1}\sum_{i=1}^{n_{l-1}}\sum_{j=1}^{n_l}\left(w_{ji}^{[l]}\right)^2}_{\text{Factor de regularizare}} \tag{6.30}$$

unde $\lambda > 0$ este coeficientul de regularizare. Ultimul termen este regularizare L_2, o sumă de pătrate de norme Frobenius peste matricele $\mathbf{W}^{[1]}, \dots, \mathbf{W}^{[L-1]}$:

$$\sum_{i=1}^{n_{l-1}}\sum_{j=1}^{n_l}\left(w_{ji}^{[l]}\right)^2 \stackrel{\text{def}}{=} \left\|\mathbf{W}^{[l]}\right\|_F^2 \tag{6.31}$$

De regulă, bibliotecile pentru algebra liniară includ implementări eficiente pentru calculul normei Frobenius. O subliniere importantă este că ponderile de bias nu sunt supuse regularizării.

6.6.1 Funcţia de cost pentru problemă de regresie

În cazul unei probleme de regresie, cea mai utilizată funcţie de eroare $J\left(\mathbf{W},\mathbf{b};\mathbf{x}^{(i)},\mathbf{d}^{(i)}\right)$ ce măsoară calitatea unei predicţii pentru perechea $\left(\mathbf{x}^{(i)},\mathbf{d}^{(i)}\right)$ este eroarea L_2 pătratică:

$$J\left(\mathbf{W},\mathbf{b};\mathbf{x}^{(i)},\mathbf{d}^{(i)}\right) = \frac{1}{2}\cdot\left\|\mathbf{d}^{(i)} - \mathbf{o}^{(i)}\right\|^2 \tag{6.32}$$

unde $\|\mathbf{v}\|$ este norma L_2 a vectorului $\mathbf{v} = (v_1, \dots, v_k)^t$:

$$\|\mathbf{v}\| = \sqrt{\sum_{i=1}^{k} v_i^2}$$

În acest caz funcţia de eroare pentru un set de date cu p perechi $(\mathbf{x}^{(i)}, \mathbf{y}^{[i]})$ devine:

$$J(\mathbf{W},\mathbf{b}) = \left[\frac{1}{2p}\sum_{i=1}^{p}\left\|\mathbf{d}^{(i)} - \mathbf{o}^{(i)}\right\|^2\right] + \frac{\lambda}{2}\sum_{l=1}^{L-1}\left\|\mathbf{W}^{[l]}\right\|_F^2 \tag{6.33}$$

6.6.2 Funcţia de cost pentru discriminarea a două clase

Dacă problema este de discriminare de două clase, atunci stratul de ieşire conţine un singur neuron, cu funcţia de activare sigmoidă logistică – idee preluată de la regresia logistică binară. Funcţia de eroare pentru o singură pereche $\left(\mathbf{x}^{(i)},\mathbf{d}^{(i)}\right)$, $J(\mathbf{W},\mathbf{b};\mathbf{x}^{(i)},\mathbf{d}^{(i)})$ este:

$$J(\mathbf{W},\mathbf{b};\mathbf{x}^{(i)},\mathbf{d}^{(i)}) = d^{(i)}\ln o^{(i)} + \left(1 - d^{(i)}\right)\ln\left(1 - o^{(i)}\right) \tag{6.34}$$

deci funcţia de eroare totală va fi:

$$J(\mathbf{W}, \mathbf{b}) = \sum_{i=1}^{p} \left[d^{(i)} \ln o^{(i)} + \left(1 - d^{(i)]}\right) \ln \left(1 - o^{(i)}\right) \right] + \frac{\lambda}{2} \sum_{l=1}^{L-1} \left\| \mathbf{W}^{[l]} \right\|_F^2 \tag{6.35}$$

6.6.3 Funcţia de cost pentru $m > 2$ clase independente

Dacă în stratul de ieşire se folosesc funcţii de activare sigmoidă logistică, iar ieşirile sunt independente unele de altele, atunci funcţia de eroare pentru o singură pereche $\left(\mathbf{x}^{(i)}, \mathbf{d}^{(i)}\right)$, $J(\mathbf{W}, \mathbf{b}; \mathbf{x}^{(i)}, \mathbf{d}^{(i)})$ este:

$$J(\mathbf{W}, \mathbf{b}; \mathbf{x}^{(i)}, \mathbf{d}^{(i)}) = -\sum_{j=1}^{m} \left[d_j^{(i)} \ln o_j^{(i)} + \left(1 - d_j^{(i)}\right) \ln \left(1 - o_j^{(i)}\right) \right] \tag{6.36}$$

unde pentru vectorul $\mathbf{d}^{(i)} = \left(d_j^{(1)}, d_j^{(2)}, \ldots, d_j^{(m)}\right)^t$ se foloseşte codificarea one–hot. În acest fel, funcţia totală de eroare $J(\mathbf{W}, \mathbf{b})$ calculată pentru setul de instruire devine:

$$J(\mathbf{W}, \mathbf{b}) = -\frac{1}{p} \sum_{i=1}^{p} \sum_{j=1}^{m} \left[d_j^{(i)} \ln o_j^{(i)} + \left(1 - d_j^{(i)}\right) \ln \left(1 - o_j^{(i)}\right) \right] + \frac{\lambda}{2} \sum_{l=1}^{L-1} \left\| \mathbf{W}^{[l]} \right\|_F^2 \tag{6.37}$$

6.6.4 Funcţia de cost pentru clasificare cu $m > 2$ clase

Pentru probleme de clasificare se preferă utilizare funcţiei de eroare cross–entropy iar în stratul de ieşire funcţia de activare să fie softmax. Funcţia de eroare pentru o singură pereche $\left(\mathbf{x}^{(i)}, \mathbf{d}^{(i)}\right)$, $J(\mathbf{W}, \mathbf{b}; \mathbf{x}^{(i)}, \mathbf{d}^{(i)})$ este:

$$J(\mathbf{W}, \mathbf{b}; \mathbf{x}^{(i)}, \mathbf{d}^{(i)}) = -\sum_{j=1}^{m} d_j^{(i)} \log o_j^{(i)} \tag{6.38}$$

unde folosim, ca mai sus, codificarea one–hot. Eroarea totală este:

$$J(\mathbf{W}, \mathbf{b}) = -\frac{1}{p} \sum_{i=1}^{p} \sum_{j=1}^{m} d_j^{(i)} \log o_j^{(i)} + \frac{\lambda}{2} \sum_{l=1}^{L-1} \left\| \mathbf{W}^{[l]} \right\|_F^2 \tag{6.39}$$

6.7 Iniţializarea ponderilor reţelei

Valorile iniţiale ale ponderilor **W** şi **b** sunt setate aleator, în jurul lui zero. Este necesar ca valorile ponderilor să nu fie toate egale; dacă ar fi toate egale, fiecare neuron ar avea exact aceeaşi valoare de activare, deoarece: fiecare neuron e legat la exact aceleaşi intrări ca şi ceilaţi din stratul său; mai departe, dacă ponderile cu care se înmulţesc intrările sunt egale, atunci valoarea de activare a fiecărui neuron de pe acel strat e aceeaşi (ponderea constantă folosită se dă factor comun); argumentul e valabil începând cu primul strat ascuns. Am ajunge deci ca neuronii de pe acelaşi strat să calculeze exact aceleaşi valori, ceea ce e inutil, iar pentru stratul de ieşire s–ar prezice mereu aceleaşi valori. Efectul de simetrie obţinut cu ponderi egale în $\mathbf{W}$ se elimină prin iniţializare cu numere aleatoare. În ce priveşte ponderile de bias – din $\mathbf{b}$ – ele se iniţializează cu 0; se consideră că iniţializarea aleatoare a ponderilor $\mathbf{W}$ este suficientă pentru "spargerea simetriei".

Strategii rafinate de inţializare pentru ponderile din $\mathbf{W}$ sunt cele propuse de Xavier Glorot *et al.* [11] şi He *et al.* [10].

Pentru arhitecturile de tip deep learning se preferă o preantrenare nesupervizată a ponderilor [8] sau preluarea unor ponderi care au fost antrenate pentru un set de date (o problemă) similară cu cea curentă – transfer de învăţare[7].

6.8 Algoritmul backpropagation

Se doreşte modificarea atât a ponderilor din matricele $\mathbf{W}^{[l]}$ cât şi a coeficienţilor de bias $\mathbf{b}^{[l]}$ astfel încât valoarea funcţiei de eroare $J(\mathbf{W}, \mathbf{b})$ să scadă. Se va folosi algoritmul de căutare după direcţia gradientului[8], în care modificarea unei ponderi $w_{ij}^{[l]}$ se efectuează astfel:

$$w_{ij}^{[l]} = w_{ij}^{[l]} - \alpha \frac{\partial J}{\partial w_{ij}^{[l]}}(\mathbf{W}, \mathbf{b}) \tag{6.40}$$

Ponderile de bias $b_{ij}^{[l]}$ se modifică similar:

$$b_{i}^{[l]} = b_{i}^{[l]} - \alpha \frac{\partial J(\mathbf{W}, \mathbf{b})}{\partial b_{i}^{[l]}}(\mathbf{W}, \mathbf{b}) \tag{6.41}$$

unde pentru ambele ecuaţii de mai sus $\alpha > 0$ este rata de învăţare.

Avem trei variante de lucru pentru modificarea ponderilor:

1. **varianta stochastic gradient descent:** pentru fiecare pereche din setul de instruire $\left(\mathbf{x}^{(i)}, \mathbf{d}^{(i)}\right)$ se calculează valoarea funcţiei de eroare $J\left(\mathbf{W}, \mathbf{b}; \mathbf{x}^{(i)}, \mathbf{d}^{(i)}\right)$, se calculează şi se aplică modificările pentru ponderile $w_{ij}^{[l]}$ şi $b_i^{[l]}$; următoarea pereche de instruire foloseşte valorile de ponderi modificate la acest pas;

[7] Eng: transfer learning
[8] Eng: gradient descent

2. **varianta off–line** sau **batch**: se calculează modificările de ponderi $w_{ij}^{[l]}$ şi $b_i^{[l]}$ care trebuie efectuate pentru fiecare pereche de vectori din setul de instruire; la final se calculează media tuturor acestor modificări şi se actualizează fiecare pondere $w_{ij}^{[l]}$ şi $b_i^{[l]}$ cu media corespunzătoare. Algoritmul dat mai jos implementează această versiune.

3. **varianta minibatch:** se împarte setul de instruire $\mathcal{S}$ în subseturi disjuncte (*minibatches*) de exemplu, de câte 100 de perechi $\left(\mathbf{x}^{(i)}, \mathbf{d}^{(i)}\right)$; pentru fiecare minibatch se calculează media modificărilor datorate valorilor din minibatch; se modifică ponderile $w_{ij}^{[l]}$ şi $b_i^{[l]}$ cu media pe minibatch, apoi se trece la următorul minibatch. Este o variantă intermediară între stochastic gradient descent – modificarea se face imediat după fiecare pereche din $\mathcal{S}$ – şi cea batch – în care modificarea ponderilor se face doar după ce se procesează tot setul $\mathcal{S}$; în practică este cea mai folosită strategie.

În toate cazurile de mai sus: o trecere completă peste setul de instruire se numeşte epocă. Se execută mai multe epoci de instruire. Numărul epocilor de instruire poate fi:

- dat apriori, de exemplu 200;
- determinat prin urmărirea valorii funcţiei de eroare peste un set de validare, un set disjunct faţă de setul de instruire $\mathcal{S}$. Dacă se constată că eroarea pe setul de validare începe să crească în timp ce eroarea pe setul de instruire continuă să scadă, atunci se decide încetarea instruirii – a se vedea figura 6.7
- se urmăreşte evoluţia normelor gradienţilor: dacă aceste norme sunt aproape de zero, înseamnă că s–a ajuns într-un punct de minim (local sau global) şi ponderile nu vor mai fi modificate semnificativ.

Vom prezenta varianta de instruire *batch*, întrucât poate fi uşor adaptată la minibatch sau stochastic gradient descent. Trecerea de la ecuaţiile (6.33) şi (6.39) la cazul în care se face antrenarea pe minibatch-uri este imediată: însumarea de p termeni din setul de instruire se substituie cu însumare peste termenii care compun acel minibatch. Pentru stochastic gradient descent avem numitorul 1.

Profitând de faptul că funcţia de eroare este o sumă de termeni şi derivata unei sume de funcţii este suma derivatelor, avem:

$$\frac{\partial J}{\partial w_{ij}^{[l]}}(\mathbf{W}, \mathbf{b}) = \left(\frac{1}{p}\sum_{k=1}^{p}\frac{\partial J}{\partial w_{ij}^{[l]}}\left(\mathbf{W}, \mathbf{b}; \mathbf{x}^{(k)}, \mathbf{d}^{(k)}\right)\right) + \lambda w_{ij}^{[l]} \tag{6.42}$$

respectiv pentru ponderile de bias

$$\frac{\partial J}{\partial b_i^{[l]}}(\mathbf{W}, \mathbf{b}) = \frac{1}{p}\sum_{k=1}^{p}\frac{\partial J}{\partial b_i^{[l]}}\left(\mathbf{W}, \mathbf{b}; \mathbf{x}^{(k)}, \mathbf{d}^{(k)}\right) \tag{6.43}$$

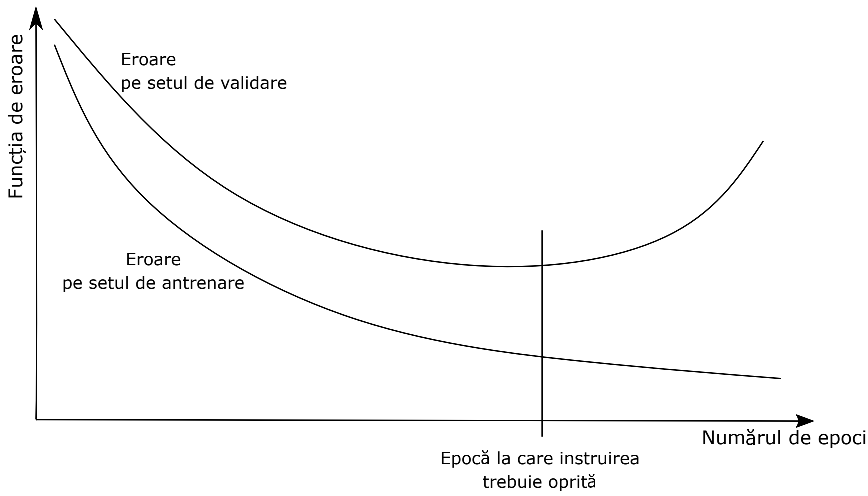


Figura 6.7: Evoluţia valorilor funcţiei de eroare pentru setul de antrenare, respectiv cel de validare. Dacă se contină antrenarea, eroare pe setul de antrenare scade, dar pentru setul de testare începe să crească de la o anumită epocă. Se recomandă oprirea instruirii dacă eroarea pe setul de validare începe să crească.

Formele matriceale ale (6.42) şi (6.43) rezultă imediat şi sunt explicitate în algoritmul de instruire ce urmează.

Algoritmul backpropagation este cel care specifică ordinea de calculare a derivatelor parţiale. Vom folosi în cele ce urmează funcţia de eroare L_2 pătratică conform ecuaţiei (6.32); pentru funcţia de eroare cross–entropy e nevoie să se calculeze corespunzător formele derivatelor parţiale ale funcţiei de eroare – a se vedea finalul acestei secţiuni.

Algoritmul funcţionează astfel: pentru o pereche de instruire $\left(\mathbf{x}^{(i)}, \mathbf{d}^{(i)}\right)$ se face pasul de propagare înainte şi se obţine vectorul de ieşire $\mathbf{o}^{(i)}$; pentru fiecare strat de neuroni l, începând de la ultimul, se calculează un termen de eroare $\boldsymbol{\delta}^{[l]} = \frac{\partial J}{\partial \mathbf{z}^{[l]}}$ care măsoară cât de mult e "responsabil" stratul l (mai precis: fiecare neuron din strat) pentru discrepanţa dintre ieşirea $\mathbf{o}^{(i)}$ şi valoarea dorită $\mathbf{d}^{(i)}$.

Înainte de detalia algoritmul de instruire a reţelei MLP, este nevoie să introducem produsul Hadamard al două matrice; produsul se notează cu $\odot$ şi se aplică pentru două matrice care au acelaşi număr de linii şi respectiv acelaşi număr de coloane: dacă $A = (a_{ij})_{1\leq i\leq q, 1\leq j\leq r}$ şi $B = (b_{ij})_{1\leq i\leq q, 1\leq j\leq r}$ sunt cele două matrice, atunci matricea produs Hadamard $C = A \odot B = (c_{ij})_{1\leq i\leq q, 1\leq j\leq r}$ are elementele:

$$c_{ij} = a_{ij} \cdot b_{ij} \tag{6.44}$$

Algoritmul backpropagation detaliat – varianta batch – este:

1. Iniţializează valorile $\Delta \mathbf{W}^{[l]}$, $\Delta \mathbf{b}^{[l]}$ cu matrice nule, pentru $l = 1, \ldots, L-1$:

$$\Delta \mathbf{W}^{[l]} = \mathbf{0}_{n_l \times n_{l-1}} \tag{6.45}$$
$$\Delta \mathbf{b}^{[l]} = \mathbf{0}_{n_l}, \text{vector coloană} \tag{6.46}$$

2. Pentru fiecare pereche $\left(\mathbf{x}^{(i)}, \mathbf{d}^{(i)}\right)$ calculează corecţia pentru ponderi şi ponderile de bias:

   2.1. Efectuează pasul de propagare înainte, conform secţiunii 6.5, şi obţine ieşirea estimată $\mathbf{o}^{(i)}$

   2.2. Pentru stratul de ieşire calculează semnalul de eroare:

$$\boldsymbol{\delta}^{[L-1]} = -\left(\mathbf{d}^{(i)} - \mathbf{o}^{(i)}\right) \odot f^{[L-1]'}\left(\mathbf{z}^{[L-1]}\right) \tag{6.47}$$

   unde am presupus că funcţia $f^{[L-1]'}$ se vectorizează peste vectorul $\mathbf{z}^{[L-1]}$.

   2.3. Pentru straturile $l = L-2, \ldots 1$ se calculează semnalul de eroare:

$$\boldsymbol{\delta}^{[l]} = \left[\left(\mathbf{W}^{[l+1]}\right)^t \cdot \boldsymbol{\delta}^{[l+1]}\right] \odot f^{[l]'}\left(\mathbf{z}^{[l]}\right) \tag{6.48}$$

   2.4. Calculează derivatele parţiale dorite, pentru $l = 1, \ldots, L-1$:

$$\frac{\partial J}{\partial \mathbf{W}^{[l]}}(\mathbf{W}, \mathbf{b}; \mathbf{x}^{(i)}, \mathbf{d}^{(i)}) = \boldsymbol{\delta}^{[l]} \cdot \left(\mathbf{a}^{[l-1]}\right)^t \tag{6.49}$$

   respectiv pentru ponderile de bias:

$$\frac{\partial J}{\partial \mathbf{b}^{[l]}}(\mathbf{W}, \mathbf{b}; \mathbf{x}^{(i)}, \mathbf{d}^{(i)}) = \boldsymbol{\delta}^{[l]} \tag{6.50}$$

   2.5. Acumulează semnalul de corecţie, pentru $l = 1, \ldots, L-1$:

$$\Delta \mathbf{W}^{[l]} = \Delta \mathbf{W}^{[l]} + \frac{\partial J}{\partial \mathbf{W}^{[l]}}(\mathbf{W}, \mathbf{b}; \mathbf{x}^{(i)}, \mathbf{d}^{(i)}) \tag{6.51}$$

$$\Delta \mathbf{b}^{[l]} = \Delta \mathbf{b}^{[l]} + \frac{\partial J}{\partial \mathbf{b}^{[l]}}(\mathbf{W}, \mathbf{b}; \mathbf{x}^{(i)}, \mathbf{d}^{(i)}) \tag{6.52}$$

3. După ce toate perechile din setul de instruire au fost considerate, modifică valorile ponderilor şi coeficienţii de bias, pentru $l = 1, \ldots L-1$:

$$\mathbf{W}^{[l]} = \mathbf{W}^{[l]} - \alpha \left[\left(\frac{1}{p}\Delta \mathbf{W}^{[l]}\right) + \lambda \mathbf{W}^{[l]}\right] \tag{6.53}$$

$$\mathbf{b}^{[l]} = \mathbf{b}^{[l]} - \alpha \left[\left(\frac{1}{p}\Delta \mathbf{b}^{[l]}\right)\right] \tag{6.54}$$

4. Repetare: se repetă de la pasul 1 până când eroarea $J(\mathbf{W}, \mathbf{b})$ scade sub un prag E_{max}, sau diferenţa absolută dintre două valori succesive ale funcţiei de eroare este sub un prag ε, sau se atinge un număr maxim de epoci.

Pentru cazul în care funcţia de eroare este cross–entropy (6.39) în loc de eroarea L_2 pătratică, se arată că pentru perechea $(\mathbf{x^{(i)}}, \mathbf{d^{(i)}})$:

$$\boldsymbol{\delta}^{[L-1]} = \mathbf{a}^{[L-1]} - \mathbf{d}^{(i)} \tag{6.55}$$

substituind formula de calcul (6.47). Utilizarea funcţiei de eroare cross entropy duce la o viteză mai mare de învăţare pentru probleme de clasificare decât dacă se foloseşte L_2 pătratică [9].

6.9 Justificarea matematică a algoritmului de backpropagation

Figura 6.8 prezintă, pentru o reţea neurală cu un strat ascuns, propagarea înainte şi înapoi prin reţea. Propagarea înainte preia o pereche de vectori $(\mathbf{x}, \mathbf{o})$. Se calculează succesiv:

- vectorul de stări pentru neuronii din stratul ascuns:

$$\mathbf{z}^{[1]} = \mathbf{W}^{[1]}\mathbf{x} + \mathbf{b}^{[1]} \tag{6.56}$$

- vectorul de valori de ieşire din stratul ascuns:

$$\mathbf{a}^{[1]} = f^{[1]}(\mathbf{z}^{[1]}) \tag{6.57}$$

- vectorul de stări pentru neuronii din stratul de ieşire:

$$\mathbf{z}^{[2]} = \mathbf{W}^{[2]}\mathbf{a}^{[1]} + \mathbf{b}^{[2]} \tag{6.58}$$

- vectorul de valori de ieşire din reţea:

$$\mathbf{o} = \mathbf{a}^{[2]} = f^{[2]}(\mathbf{z}^{[2]}) \tag{6.59}$$

- Valoarea de eroare $J\left(\mathbf{W}, \mathbf{b}; \mathbf{x}^{(k)}, \mathbf{d}^{(k)}\right)$, notată în figura 6.8 cu $J(\mathbf{a}^2, \mathbf{y})$.

Avem nevoie de derivatele parţiale:

$$\frac{\partial J}{\partial \mathbf{W}^{[1]}}, \frac{\partial J}{\partial \mathbf{b}^{[1]}}, \frac{\partial J}{\partial \mathbf{W}^{[2]}}, \frac{\partial J}{\partial \mathbf{b}^{[2]}} \tag{6.60}$$

pentru a face modificarea ponderilor.

Din motive de eficientizare a calculelor, le vom calcula în ordinea descrescătoare a indicilor superiori. Reamintim formula de derivare a funcţiilor compuse: dacă avem compunerea de funcţii $h = f(g(\cdot))$, atunci derivata lui fh în raport cu x este:

$$\frac{dh}{dx} = \frac{df}{dg}\frac{dg}{dx} \tag{6.61}$$

Noi vom folosi această proprietate pentru derivate parţiale.

Pentru calculul derivatelor parţiale $\frac{\partial J}{\partial \mathbf{W}^{[2]}}$ avem:

$$\frac{\partial J}{\partial \mathbf{W}^{[2]}} = \frac{\partial J}{\partial \mathbf{z}^{[2]}}\frac{\partial \mathbf{z}^{[2]}}{\partial \mathbf{W}^{[2]}} \tag{6.62}$$

Derivata parţială $\frac{\partial \mathbf{z}^{[2]}}{\partial \mathbf{W}^{[2]}}$ se calculează simplu, ţinând cont de forma liniară din ec. (6.58):

$$\frac{\partial \mathbf{z}^{[2]}}{\partial \mathbf{W}^{[2]}} = \mathbf{a}^{[1]} \tag{6.63}$$

deci (6.62) devine:

$$\frac{\partial J}{\partial \mathbf{W}^{[2]}} = \frac{\partial J}{\partial \mathbf{z}^{[2]}}\mathbf{a}^{[1]} \tag{6.64}$$

Pentru derivata parţială $\frac{\partial J}{\partial \mathbf{b}^{[2]}}$ avem:

$$\frac{\partial J}{\partial \mathbf{b}^{[2]}} = \frac{\partial J}{\partial \mathbf{z}^{[2]}}\frac{\partial \mathbf{z}^{[2]}}{\partial \mathbf{b}^{[2]}} \tag{6.65}$$

unde al doilea termen din partea dreaptă se calculează simplu, folosind definiţia din ec. (6.58):

$$\frac{\partial \mathbf{z}^{[2]}}{\partial \mathbf{b}^{[2]}} = 1 \tag{6.66}$$

deci (6.65) devine:

$$\frac{\partial J}{\partial \mathbf{b}^{[2]}} = \frac{\partial J}{\partial \mathbf{z}^{[2]}} \tag{6.67}$$

Observăm că în ambele ecuaţii (6.64) şi (6.67) apare cantitatea $\frac{\partial J}{\partial \mathbf{z}^{[2]}}$ care se calculează simplu, astfel:

$$\frac{\partial J}{\partial \mathbf{z}^{[2]}} = \frac{\partial J}{\partial \mathbf{a}^{[2]}}\frac{\partial \mathbf{a}^{[2]}}{\partial \mathbf{z}^{[2]}} = \frac{\partial J}{\partial \mathbf{a}^{[2]}} \cdot f^{(2)\prime}(\mathbf{z}^{[2]}) \tag{6.68}$$

Termenul $\frac{\partial J}{\partial \mathbf{z}^{[2]}}$ este semnalul de eroare pentru stratul 2. Termenul $\frac{\partial J}{\partial \mathbf{a}^{[2]}}$, notat cu $da^{[2]}$ în figura 6.8 depinde de forma concretă a funcţiei de eroare J.

Pentru derivatele parţiale după $\mathbf{W}^{[1]}$ şi $\mathbf{b}^{[1]}$ procedăm similar:

$$\frac{\partial J}{\partial \mathbf{W}^{[1]}} = \frac{\partial J}{\partial \mathbf{z}^{[1]}}\frac{\partial \mathbf{z}^{[1]}}{\partial \mathbf{W}^{[1]}} = \frac{\partial J}{\partial \mathbf{z}^{[1]}}\mathbf{x} \tag{6.69}$$

respectiv

$$\frac{\partial J}{\partial \mathbf{b}^{[1]}} = \frac{\partial J}{\partial \mathbf{z}^{[1]}} \frac{\partial \mathbf{z}^{[1]}}{\partial \mathbf{b}^{[1]}} = \frac{\partial J}{\partial \mathbf{z}^{[1]}} \tag{6.70}$$

unde semnalul de eroare $\frac{\partial J}{\partial \mathbf{z}^{[1]}}$ pentru stratul 1 este

$$\frac{\partial J}{\partial \mathbf{z}^{[1]}} = \frac{\partial J}{\partial \mathbf{a}^{[2]}} \frac{\partial \mathbf{a}^{[2]}}{\partial \mathbf{z}^{[2]}} = \frac{\partial J}{\partial \mathbf{z}^{[3]}} \cdot \frac{\partial \mathbf{z}^{[3]}}{\partial \mathbf{a}^{[2]}} \cdot \frac{\partial \mathbf{a}^{[2]}}{\partial \mathbf{z}^{[2]}} \tag{6.71}$$

unde

$$\frac{\partial \mathbf{z}^{[3]}}{\partial \mathbf{a}^{[2]}} = \mathbf{W}^{[2]} \tag{6.72}$$

şi

$$\frac{\partial \mathbf{a}^{[2]}}{\partial \mathbf{z}^{[2]}} = f^{(2)\prime}(\mathbf{z}^{[2]}) \tag{6.73}$$

iar în $\frac{\partial J}{\partial \mathbf{z}^{[2]}}$ recunoaştem un semnal de eroare calculat anterior în ecuaţia (6.68).

6.10 Utilizarea reţelei

După ce se face antrenarea reţelei, ea se poate folosi pentru a face predicţii pentru date din setul de testare $\mathcal{T} = \left\{\mathbf{x}^{(j)} | 1 \leq j \leq q\right\}$. Fiecare vector din $\mathcal{T}$ este trecut prin reţea, conform pasului de propagare înainte şi se obţin valori de ieşire estimate (predicţii) $\mathbf{o}^{(1)}, \ldots, \mathbf{o}^{(q)}$, toate din $\mathbb{R}^m$.

Dacă valorile de ieşire sunt interpretate ca probabilităţi condiţionate, adică:

$$o_i = P\left(\text{clasa } i | \mathbf{x}\right), 1 \leq i \leq m \tag{6.74}$$

atunci clasificarea se face găsind acel indice i pentru care o_i e maxim şi acesta este indicele clasei prezise de reţeaua MLP.

6.11 Discuţii

TODO:

- problema minimelor locale
- arhitecture reţelei: număr de straturi ascunse, număr de neuroni pe strat
- dependenta de ordine
- alţi algoritmi decât SGD
- learning rate scheduling
- capacitatea de aproximare universală

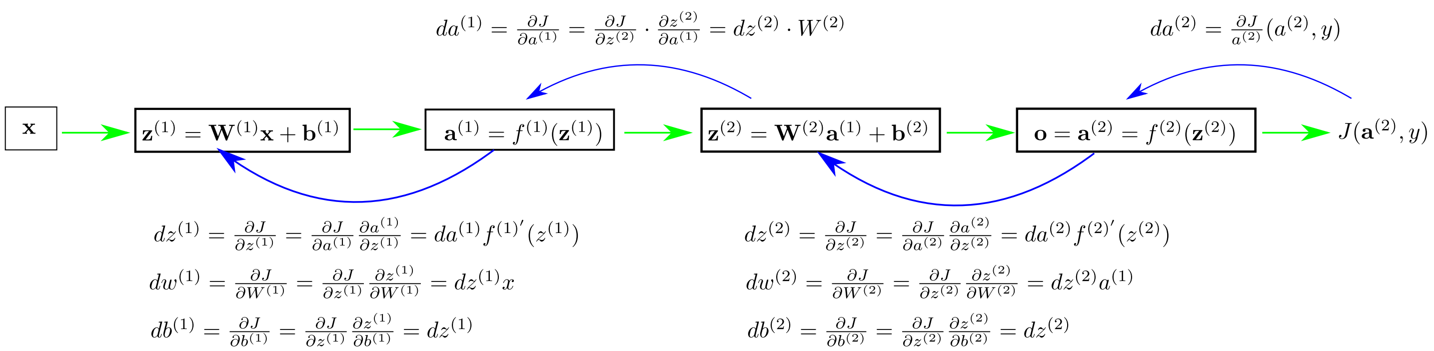


Figura 6.8: Reprezentarea procesului de propagare înainte (arcele verzi) şi înapoi (arcele albastre). Valoarea de eroare $J\left(\mathbf{a}^{[2]}, \mathbf{y}\right)$ este asociată unei perechi $\left(\mathbf{x}^{(i)}, \mathbf{y}^{[i]}\right)$.

Capitolul 7

Memorii asociative bidirecţionale

Memoriile asociative bidirecţionale (MAB) permit stocarea şi regăsirea datelor. Căutarea se face pe baza similarităţii care există între vectorul furnizat ca intrare şi ceea ce este stocat în reţea. Regăsirea se face pe baza similarităţii între vectorul furnizat iniţial şi a unui vector stocat de reţea. Se lucrează cu perechi de vectori asociaţi memoraţi de reţea; plecând de la oricare dintre ele sau de la unul similar cu ele se doreşte regăsirea celuilalt. Datele memorate sunt reprezentate în ponderile reţelei.

Instruirea poate fi atât supervizată, cât şi nesupervizată. Spre deosebire de modelele anterioare, MAB–urile nu se folosesc pentru regresie, clasificare sau estimare de probabilitate condiţionată. Este sunt utile pentru regăsirea pe bază de conţinut, reconstituire şi corectare de date.

Întrucât memoria regăseşte datele de instruire pe baza similarităţii, este necesară o discuţie despre distanţe şi spaţiul din care fac parte datele.

7.1 Distanţa Hamming

Fie $\mathbf{x} = (x_1, \dots, x_n)^t$ şi $\mathbf{y} = (y_1, \dots, y_n)^t$ doi vectori n-dimensionali din spaţiul Euclidian având restricţiile $x_i, y_i \in \{-1, +1\}$, $i = 1, \dots, n$. Cea mai frecvent utilizată metrică, distanţa Euclidiană, dintre doi vectori este:

$$d_E(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \tag{7.1}$$

Având în vedere valorile pe care le pot lua componentele vectorilor $\mathbf{x}$ şi $\mathbf{y}$, avem că:

$$(x_i - y_i)^2 = \begin{cases} 0 & \text{dacă } x_i = y_i \\ (\pm 2)^2 = 4 & \text{dacă } x_i \neq y_i \end{cases} \tag{7.2}$$

deci distanţa Euclidiană este $d_E(\mathbf{x}, \mathbf{y}) = \sqrt{4 \cdot diferente(\mathbf{x}, \mathbf{y})}$ unde prin $diferente(\mathbf{x}, \mathbf{y})$ am notat numărul de componente din $\mathbf{x}$ şi $\mathbf{y}$ care diferă pentru poziţii de acelaşi indice. Pentru doi vectori $\mathbf{x}$ şi $\mathbf{y}$ ca mai sus se defineşte funcţia de *distanţă Hamming* d_H ca fiind tocmai numărul de diferenţe de pe poziţiile corespunzătoare:

$$d_H(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{n} I(x_i \neq y_i) \tag{7.3}$$

unde $I(\cdot)$ este funcţia indicator (3.23) de la pagina 42.

Există deci relaţia:

$$d_E(\mathbf{x}, \mathbf{y}) = 2\sqrt{d_H(\mathbf{x}, \mathbf{y})} \tag{7.4}$$

Considerăm hipercubul n-dimensional centrat în origine cu latura de lungime 2:

$$\mathbf{H}^n = \left\{\mathbf{x} = (x_1, x_2, \dots, x_n)^t \in \mathbb{R}^n, x_i \in \{-1, +1\}\right\} = \{-1, +1\}^n \tag{7.5}$$

$\mathbf{H}^n$ se mai numeşte şi cub Hamming.

7.2 Asociatori

Considerăm mulţimea de p perechi de vectori de instruire:

$$\mathcal{S} = \left\{\left(\mathbf{x}^{(1)}, \mathbf{y}^{(1)}\right), \dots, \left(\mathbf{x}^{(p)}, \mathbf{y}^{(p)}\right)\right\} \tag{7.6}$$

unde $\mathbf{x}^{(i)} \in \mathbb{R}^n$, $\mathbf{y}^{(i)} \in \mathbb{R}^m$, $1 \le i \le p$. Există trei tipuri de memorii asociative:

1. *Memorii heteroasociative*, implementând o funcţie $\Phi : \mathbb{R}^n \to \mathbb{R}^m$ cu proprietatea că $\Phi\left(\mathbf{x}^{(i)}\right) = \mathbf{y}^{(i)}$, $1 \le i \le p$. În plus, cerem ca dacă un vector $\mathbf{x} \in \mathbb{R}^n$ este cel mai apropiat de un exemplar $\mathbf{x}^{(i)}$ din $\mathcal{S}$ $(1 \le i \le p)$, atunci $\Phi(\mathbf{x}) = \Phi\left(\mathbf{x}^{(i)}\right) = \mathbf{y}^{(i)}$. Apropierea se consideră în sensul unei distanţe convenabil alese - motiv pentru care s-a discutat în prima parte a cursului despre distanţe.

2. *Memorii interpolative*, implementând o funcţie $\Phi : \mathbb{R}^n \to \mathbb{R}^m$ astfel încât $\Phi(\mathbf{x}^{(i)}) = \mathbf{y}^{(i)}$, $1 \le i \le p$. În plus, dacă vectorul $\mathbf{x} \in \mathbb{R}^n$ este $\mathbf{x} = \mathbf{x}^{(i)} + \mathbf{d}$ unde $\mathbf{x}^{(i)}$ e cel mai apropiat de $\mathbf{x}$, atunci ieşirea memoriei este:
$$\Phi(\mathbf{x}) = \Phi\left(\mathbf{x}^{(i)} + \mathbf{d}\right) = \mathbf{y}^{(i)} + \mathbf{e},\ \mathbf{e} \in \mathbb{R}^m \tag{7.7}$$

3. *Memorie autoasociativă:* dacă $\mathbf{y}^{(i)} = \mathbf{x}^{(i)}$, $1 \leq i \leq p$, atunci o memorie autoasociativă trebuie să respecte proprietăţile date de memoria heteroasociativă: $\Phi\left(\mathbf{x}^{(i)}\right) = \mathbf{x}^{(i)}$, $1 \leq i \leq p$ şi dacă $\mathbf{x}$ este cel mai apropiat de un exemplar $\mathbf{x}^{(i)}$, atunci $\Phi(\mathbf{x}) = \Phi\left(\mathbf{x}^{(i)}\right) = \mathbf{x}^{(i)}$.

Pentru cazul în care setul de vectori $\left\{\mathbf{x}^{(i)} | 1 \leq i \leq p\right\}$ este ortonormat[1] putem construi simplu un asociator interpolativ: Definim funcţia Φ ca fiind:

$$\Phi(\mathbf{x}) = \left(\mathbf{y}^{(1)}\mathbf{x}^{(1)t} + \cdots + \mathbf{y}^{(p)}\mathbf{x}^{(p)t}\right)\mathbf{x} \tag{7.8}$$

Avem că:

$$\Phi(\mathbf{x}^{(i)}) = \left(\mathbf{y}^{(1)}\mathbf{x}^{(1)t} + \cdots + \mathbf{y}^{(p)}\mathbf{x}^{(p)t}\right)\mathbf{x}^{(i)} = \sum_{j=1}^{p}\left(\left(\mathbf{y}^{(j)}\mathbf{x}^{(j)t}\right)\mathbf{x}^{(i)}\right) =$$
$$= \sum_{j=1}^{p}\left(\mathbf{y}^{(j)}\left(\mathbf{x}^{(j)t}\mathbf{x}^{(i)}\right)\right) = \sum_{j=1}^{p}\left(\mathbf{y}^{(j)} \cdot I(i=j)\right) = \mathbf{y}^{(i)}. \tag{7.9}$$

Dacă un argment $\mathbf{x} \in \mathbb{R}^n$ are forma $\mathbf{x} = \mathbf{x}^{(i)} + \mathbf{d}$, atunci:

$$\Phi(\mathbf{x}) = \Phi\left(\mathbf{x}^{(i)} + \mathbf{d}\right) = \mathbf{y}^{(i)} + \mathbf{e} \tag{7.10}$$

unde

$$\mathbf{e} = \left(\sum_{j=1}^{p}\mathbf{y}^{(j)}\mathbf{x}^{(j)t}\right) \cdot \mathbf{d} \tag{7.11}$$

7.3 Memoria asociativă bidirecţională

O memorie asociativă bidirecţională (MAB) este o memorie heteroasociativă constând în două straturi de elemente de procesare (noduri) care sunt interconectate. Elementele pot sau nu să aibă legături cu ele însele (bucle). O reprezentare este dată în figura 7.1. Valorile $\mathbf{x}$ sunt din $\mathbf{H}^n$ iar $\mathbf{y}$ din $\mathbf{H}^m$. Între noduri există legături cu diferite ponderi.

Spre deosebire de alte tipuri de reţele neurale, ponderile pot fi determinate dacă se cunoaşte de dinainte setul de exemplare ce trebuie memorat. Conexiunile sunt bidirecţionale: putem furniza ca intrare o valoare în stratul $\mathbf{x}$ iar ieşirea să fie dată de stratul $\mathbf{y}$ sau invers.

Pentru construirea matricii ponderilor se poate folosi o idee similară cu cea de la memoria interpolativă:

$$\mathbf{W} = \mathbf{y}^{(1)}\mathbf{x}^{(1)t} + \cdots + \mathbf{y}^{(p)}\mathbf{x}^{(p)t}. \tag{7.12}$$

[1] Vectorii $\left\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(p)} | 1 \leq i \leq p\right\}$ sunt ortonormaţi dacă $\mathbf{x}^{(i)t} \cdot \mathbf{x}^{(j)} = I(i=j)$, $1 \leq i, j \leq p$. Dintr-un set de vectori liniar independenţi putem obţine întotdeauna un sistem de vectori ortonormaţi prin procedeul Gram-Schmidt de ortonormare. Pentru a reduce din efectul erorilor de rotunjire, se poate folosi procedeul Gram-Schmidt modificat.

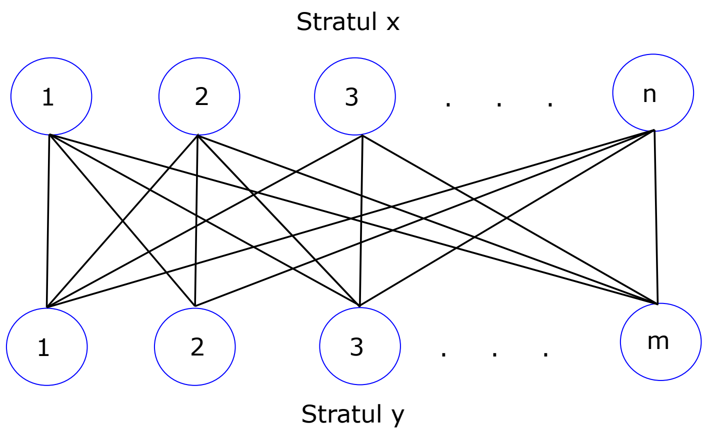


Figura 7.1: Arhitectura unei memorii asociative bidirecţionale

Matricea $\mathbf{W} = (w_{ij})$, $1 \le i \le m$, $1 \le j \le n$ dă ponderile legăturilor de la stratul $\mathbf{x}$ la stratul $\mathbf{y}$. Matricea ponderilor în sens invers este $\mathbf{W}^t$. Memoria poate deveni autoasociativă prin stabilirea lui $\mathbf{W}$ ca fiind:

$$\mathbf{W} = \mathbf{x}^{(1)}\mathbf{x}^{(1)t} + \cdots + \mathbf{x}^{(p)}\mathbf{x}^{(p)t} \tag{7.13}$$

Odată matricea de ponderi contruită, se poate utiliza MAB pentru regăsirea datelor stocate prin furnizarea unor date cheie, suficient de apropiate de cele din setul de instruire. Vom vedea că această intrare poate fi obţinută prin perturbarea unei valori din setul de instruire, iar MAB poate încă să determine cheia originară şi valoarea asociată ei.

Paşii de lucru sunt următorii:

1. se aplică perechea iniţială de vectori $(\mathbf{x}, \mathbf{y})$ celor două straturi de neuroni;

2. se propagă informaţia de la stratul $\mathbf{x}$ la stratul $\mathbf{y}$ şi se modifică valorile din stratul $\mathbf{y}$;

3. se propagă informaţia de la $\mathbf{y}$ la $\mathbf{x}$ şi se modifică valorile din stratul $\mathbf{x}$;

4. se repetă paşii 2 şi 3 până când nu mai apare nicio modificare în nodurile celor două straturi.

Se poate ca datele să înceapă să se propage de la stratul $\mathbf{y}$ la stratul $\mathbf{x}$. Plimbarea informaţiei în ambele sensuri dă natura bidirecţională a reţelei. Când reţeaua se stabilizează, de regulă se regăseşte în stratul $\mathbf{x}$ valoarea $\mathbf{x}^{(i)}$ care este cea mai apropiată de $\mathbf{x}$ relativ la distanţa Hamming şi valoarea $\mathbf{y}^{(i)}$ asociată cu $\mathbf{x}^{(i)}$ — sau complementele acestora, a se vedea exemplele următoare.

Procesarea care se face în momentul transmiterii informaţiei de la stratul $\mathbf{x}$ la stratul $\mathbf{y}$ este dată de ecuaţia:

$$\mathbf{net}^y = \mathbf{W} \cdot \mathbf{x} \tag{7.14}$$

sau pe componente:

$$net_i^y = \sum_{j=1}^{n} w_{ij} x_j, \ 1 \leq i \leq m \tag{7.15}$$

unde $\mathbf{net}^y$ este vectorul de stare pentru stratul $\mathbf{y}$. Pentru transmiterea în sens invers are loc un proces asemănător:

$$\mathbf{net}^x = \mathbf{W}^t \mathbf{y} \tag{7.16}$$

sau pe componente:

$$net_i^x = \sum_{j=1}^{m} w_{ji} \cdot y_j, \ 1 \leq i \leq n \tag{7.17}$$

Valoarea de ieşire a unui neuron depinde de intrări şi de valoarea lui curentă. Mai clar, valoarea de la momentul $t+1$ pentru nodul y_i este dată de:

$$y_i(t+1) = \begin{cases} +1, & \text{dacă } net_i^y > 0 \\ y_i(t), & \text{dacă } net_i^y = 0 \\ -1, & \text{dacă } net_i^y < 0 \end{cases} \tag{7.18}$$

Valorile de ieşire pentru stratul $\mathbf{x}$ se calculează similar.

Exemplu numeric: plecăm de la perechea de vectori

$$\mathbf{x}^{(1)} = (1, -1, -1, 1, -1, 1, 1, -1, -1, 1)^t \tag{7.19}$$
$$\mathbf{x}^{(2)} = (1, 1, 1, -1, -1, -1, 1, 1, -1, -1)^t \tag{7.20}$$

cu ieşirile corespunzătoare asociate

$$\mathbf{y}^{(1)} = (1, -1, -1, -1, -1, 1)^t \tag{7.21}$$
$$\mathbf{y}^{(2)} = (1, 1, 1, 1, -1, -1)^t \tag{7.22}$$

Matricea ponderilor este:

$$\mathbf{W} = \begin{pmatrix} 2 & 0 & 0 & 0 & -2 & 0 & 2 & 0 & -2 & 0 \\ 0 & 2 & 2 & -2 & 0 & -2 & 0 & 2 & 0 & -2 \\ 0 & 2 & 2 & -2 & 0 & -2 & 0 & 2 & 0 & -2 \\ 0 & 2 & 2 & -2 & 0 & -2 & 0 & 2 & 0 & -2 \\ -2 & 0 & 0 & 0 & 2 & 0 & -2 & 0 & 2 & 0 \\ 0 & -2 & -2 & 2 & 0 & 2 & 0 & -2 & 0 & 2 \end{pmatrix} \tag{7.23}$$

Vom considera ca vector de intrare $\mathbf{x} = (-1,-1,-1,1,-1,1,1,-1,-1,1)^t$ cu vectorul $\mathbf{y}$ asociat $(1,1,1,1,-1,-1)^t$. Remarcăm că vectorii $\mathbf{x}$ şi $\mathbf{y}$ daţi nu sunt printre vectorii învăţaţi. Vectorul de ieşire $\mathbf{y}$ poate fi dat ca un vector binar bipolar cu componente aleatoare. Propagarea valorilor dinspre stratul $\mathbf{x}$ către $\mathbf{y}$ duce la determinarea valorii $\mathbf{net}^y = (4,-12,-12,-12,-4,12)^t$. Noul vector din stratul $\mathbf{y}$ este $\mathbf{y} = (1,-1,-1,-1,-1,1)^t$. Propagând înapoi către stratul $\mathbf{x}$ obţinem $\mathbf{x} = (1,-1,-1,1,-1,1,1,-1,-1,1)^t$. Propagări succesive într–un sens sau în celălalt nu duc la modificări ale valorilor din straturile $\mathbf{x}$ sau $\mathbf{y}$. Perechea de vectori la care se stabilizează reţeaua este chiar perechea $\left(\mathbf{x}^{(1)}, \mathbf{y}^{(1)}\right)$. S-a regăsit astfel o pereche de exemplare din cele cu care s–a instruit reţeaua, chiar dacă s–a plecat de la valori care nu se regăsesc printre cele învăţate.

Să considerăm situaţia în care se pleacă de la valorile iniţiale:

$$\mathbf{x} = (-1,1,1,-1,1,1,1,-1,1,-1)^t, \mathbf{y} = (-1,1,-1,1,-1,-1)^t$$

Propagând de la stratul $\mathbf{x}$ la $\mathbf{y}$, obţinem $\mathbf{y} = (-1,1,1,1,1,-1)^t$. Propagând în direcţia inversă, obţinem $\mathbf{x} = (-1,1,1,-1,1,-1,-1,1,1,-1)^t$ şi reţeaua se stabilizează. Se observă că valorile stabile $(\mathbf{x}, \mathbf{y})$ sunt chiar $(\mathbf{x}^{(1)c}, \mathbf{y}^{(1)c})$ unde $\mathbf{a}^c$ este vectorul format cu valorile complementate ce compun pe $\mathbf{a}$: $\mathbf{a}^c = -\mathbf{a}$. Aceasta este o proprietate a MAB: dacă memoria stochează perechea $(\mathbf{x}, \mathbf{y})$, atunci stochează şi perechea $(\mathbf{x}^c, \mathbf{y}^c)$ şi stabilizarea reţelei se poate face pe o astfel de pereche de complemente.

7.4 Funcţia de energie a MAB

În timpul propagării valorilor dinspre stratul $\mathbf{x}$ spre $\mathbf{y}$ sau invers, valorile din nodurile reţelei se modifică, ceea ce ne permite să vedem evoluţia stării acestora ca o funcţie de timp. Vom asocia memoriei o funcţie de energie a cărei valoare este dependentă de valorile $\mathbf{x}$ şi $\mathbf{y}$ din noduri; vrem să arătăm că funcţia de energie converge la un punct limită pe durata propagării datelor între cele două straturi. Convergenţa se traduce prin stabilizarea reţelei.

Funcţia de energie considerată este:

$$E(\mathbf{x}, \mathbf{y}) = -\mathbf{y}^t \mathbf{W} \mathbf{x} = -\sum_{i=1}^{m} \sum_{j=1}^{n} y_i w_{ij} x_j$$

Avem următoarea teoremă privitoare la comportamentul MAB pentru funcţia de energie:

Teorema 2. *1. Orice modificare a stării stratului* $\mathbf{x}$ *sau* $\mathbf{y}$ *în timpul procesării din MAB duce la scăderea lui* E*;*

2. E *este mărginită inferior de* $E_{min} = -\sum_{i,j} |w_{ij}|$*;*

3. *Dacă valoarea lui E se schimbă, atunci modificarea nu este arbitrar de mică.*

Demonstraţie. Pentru cazul în care se face propagare dinspre stratul x către stratul y, presupunem că se face o modificare pentru vectorul $\mathbf{y}$ pe o singură poziţie, fie ea l, $1 \leq l \leq m$. Energia asociată intrării curente este:

$$E = -\sum_{i=1}^{m}\sum_{j=1}^{n} y_i w_{ij} x_j = -\sum_{j=1}^{n} y_l w_{lj} x_j - \sum_{i=1, i\neq l}^{m}\sum_{j=1}^{n} y_i w_{ij} x_j \tag{7.24}$$

Dacă se face modificarea valorii y_l în y_l^{nou}, noua valoare a energiei va fi:

$$E^{nou} = -\sum_{j=1}^{n} y_l^{nou} w_{lj} x_j - \sum_{i=1, i\neq l}^{m}\sum_{j=1}^{n} y_i w_{ij} x_j \tag{7.25}$$

şi deci variaţia energiei este:

$$\Delta E = E^{nou} - E = (y_l - y_l^{nou})\sum_{j=1}^{n} w_{lj} x_j = (y_l - y_l^{nou}) net_l^y \tag{7.26}$$

Avem posibilităţile:

1. dacă $y_l = +1$, atunci $y_l^{nou} = -1$. Avem $y_l - y_l^{nou} = 2 > 0$; pe de altă parte, dacă $y_l^{nou} = -1$, asta se datorează lui $net_l^y < 0$ (a se vedea ecuaţia 7.18). Valoarea lui ΔE este produsul a doi termeni întregi nenuli şi de semn contrar şi deci este o valoare întreagă strict mai mică decât zero.

2. dacă $y_l = -1$, atunci $y_l^{nou} = +1$ şi de aici $y_l - y_l^{nou} = -2 < 0$. Pe de altă parte, trecerea de la $y_l = -1$ la $y_l^{nou} = +1$ se datorează lui $net_l^y > 0$ (ecuaţia 7.18) şi din nou ΔE este produsul a două valori întregi nenule şi de semn contrar, ca atare de valoare întreagă strict mai mică decât zero.

Situaţia în care mai mult de un termen din $\mathbf{y}^{nou}$ este modificat faţă de $\mathbf{y}$ se tratează similar, cu observaţia că scăderea lui ΔE este şi mai accentuată.

Similar se arată că modificarea stării unui neuron din stratul de intrare de asemenea scade valoarea funcţiei de energie.

Pentru cea de a doua parte a teoremei avem:

$$-E(\mathbf{x},\mathbf{y}) = \sum_{i=1}^{m}\sum_{j=1}^{n} y_i w_{ij} x_j \overset{a\leq|a|}{\leq} \left|\sum_{i=1}^{m}\sum_{j=1}^{n} y_i w_{ij} x_j\right| \overset{|a+b|\leq|a|+|b|}{\leq}$$
$$\overset{|a+b|\leq|a|+|b|}{\leq} \sum_{i=1}^{m}\sum_{j=1}^{n} |y_i|\cdot|w_{ij}|\cdot|x_j| \overset{|x_j|=|y_i|=1}{=} \sum_{i=1}^{m}\sum_{j=1}^{n} |w_{ij}|. \tag{7.27}$$

de unde $E(\mathbf{x}, \mathbf{y}) \geq -\sum\limits_{i=1}^{m} \sum\limits_{j=1}^{n} |w_{ij}|$.

Partea a treia a teoremei rezultă din cele obţinute în demonstraţia primei părţi a teoremei: dacă a apare modificare în stările neuronilor, ΔE este un număr întreg strict mai mic decât zero. $\square$

Punctele 2 şi 3 ale teoremei arată că MAB se stabilizează într-un numıar finit de paşi. Primele două puncte arată că funcţia E este de tip Lyapunov, o clasă de funcţii des folosite pentru studiul sistemelor dinamice.

Demonstraţie practică: http://facstaff.cbu.edu/~pong/ai/bam/bamapplet.html.

7.5 Comentarii

MAB este un model în care ordinea de prezentare a vectorilor în setul de instruire $\mathcal{S}$ din ecuaţia (7.6) nu e relevantă: suma pe baza căreia se creează matricea de ponderi $\mathbf{W}$ – a se vedea ecuaţia (7.12) – este o operaţie comutativă. Altfel zis, MAB este un model *order independent*.

Totodată, MAB este un model de învăţare în care creşterea sau reducerea setului de instruire $\mathcal{S}$ (uitarea) nu necesită reluarea instruirii pe întregul set de instruire:

- la adăugarea unei noi perechi de vectori $\left(\mathbf{x}^{(p+1)}, \mathbf{y}^{(p+1)}\right)$ setului de instruire, la matricea $\mathbf{W}$ formată cu p termeni se mai adună matricea $\mathbf{y}^{(p+1)} \cdot \mathbf{x}^{(p+1)t}$;
- dacă se doreşte "uitarea" perechii $\left(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}\right)$, atunci din matricea $\mathbf{W}$ se scade termenul $\mathbf{y}^{(i)} \cdot \mathbf{x}^{(i)t}$.

Capacitatea de memorare a unei memorii asociative bidirecţionale este insuficient studiată. După unii autori ea este $\min(m, n)$, alţii estimează însă că un prag superior pentru numărul de perechi de vectori care pot fi memorate ar fi $\sqrt{\min(m, n)}$.

Capitolul 8

Reţele neurale cu funcţii de activare radială

Reţelele neurale cu funcţii de activare radială se folosesc pentru probleme de clasificare, estimare de probabilitate condiţionată şi regresie. Instruirea este nesupervizată (clustering) pentru determinarea centrilor neuronilor din stratul ascuns şi supervizată pentru învăţarea ponderilor dintre stratul ascuns şi cel de ieşire.

8.1 Teorema lui Cover

Pentru cazul vectorilor ce nu pot fi separaţi liniar, perceptronul multistrat poate determina o funcţie de separare, datorită caracterului de aproximator universal. Există şi o altă variantă de rezolva problema discernerii între clase ce nu se pot separa liniar, *folosind însă un separator liniar*, ce lucrează în doi paşi:

1. mulţimea de instruire dată în spaţiul originar este transformată într–un alt spaţiu, în care, în anumite condiţii, liniar separabilitatea poate apărea cu probabilitate mare; fundamentul matematic este dat de teorema lui Cover (vedeţi mai jos);

2. prin utilizarea unui model de separare liniară (perceptron, SVM liniar, regresie logistică) se separă clasele în cel de-al doilea spaţiu.

Rezultatul se poate materializa printr–o reţea cu funcţii de activare radială, formată din 3 straturi:

- stratul de intrare alcătuit din noduri de intrare care conectează reţeaua la mediu;

- stratul de neuroni ascunşi ce aplică transformări neliniare pe datele din spaţiul de intrare. Neuronii din acest strat sunt antrenaţi prin instruire nesupervizată;

- stratul de ieşire produce o transformare liniară, iar ponderile dintre stratul ascuns şi stratul de ieşire sunt obţinute prin instruire supervizată. Acesta furnizează valoarea de ieşire pentru vectorul de intrare curent.

Următoarea teoremă arată motivul pentru care se face o transformare a datelor originare în alte date dintr-un spaţiu cu mai multe dimensiuni decât cel originar:

Teorema 3 (Cover, 1965)**.** *Printr–o transformare neliniară a unui set de date de intrare dintr-un spaţiu A într-un spaţiu B cu dimensiune mai mare, creşte probabilitatea de a obţine clase liniar separabile, dacă A nu este dens populat.*

Rezultatul este util, deoarece pentru cazuri liniar separabile, un perceptron discret poate să obţină un hiperplan de separare în timp finit; se pot folosi si alte modele de clasificatori liniari - regresie logistică, Support Vector Machines etc. Pentru a se obţine o asemenea transformare, se pleacă de la spaţiul A n dimensional în care se găsesc vectorii $\mathbf{x}_1, \ldots, \mathbf{x}_p$ şi se ajunge la un spaţiu m dimensional ($m \geq n$) prin funcţia:

$$\mathbf{x} \xrightarrow{\boldsymbol{\phi}} \boldsymbol{\phi}(\mathbf{x}) = (\varphi_1(\mathbf{x}), \ldots, \varphi_m(\mathbf{x}))^t \in \mathbb{R}^m \tag{8.1}$$

unde $\varphi_i : \mathbb{R}^n \to \mathbb{R}, i = \overline{1, m}$ sunt funcţii neliniare; în reţeaua neurală funcţia φ_i e calculată de neuronul i din stratul ascuns.

Exemplu: considerăm problema XOR, în care 4 puncte sunt asignate la două clase, astfel: punctele de coordonate $(0, 0)^t$ şi $(1, 1)^t$ aparţin unei clase, iar $(0, 1)^t$ şi $(1, 0)^t$ aparţin celeilalte clase. Se poate demonstra algebric că nu există o dreaptă în plan care să separe cele două clase de puncte. Considerăm funcţiile $\varphi_1, \varphi_2 : \mathbb{R}^2 \to \mathbb{R}$

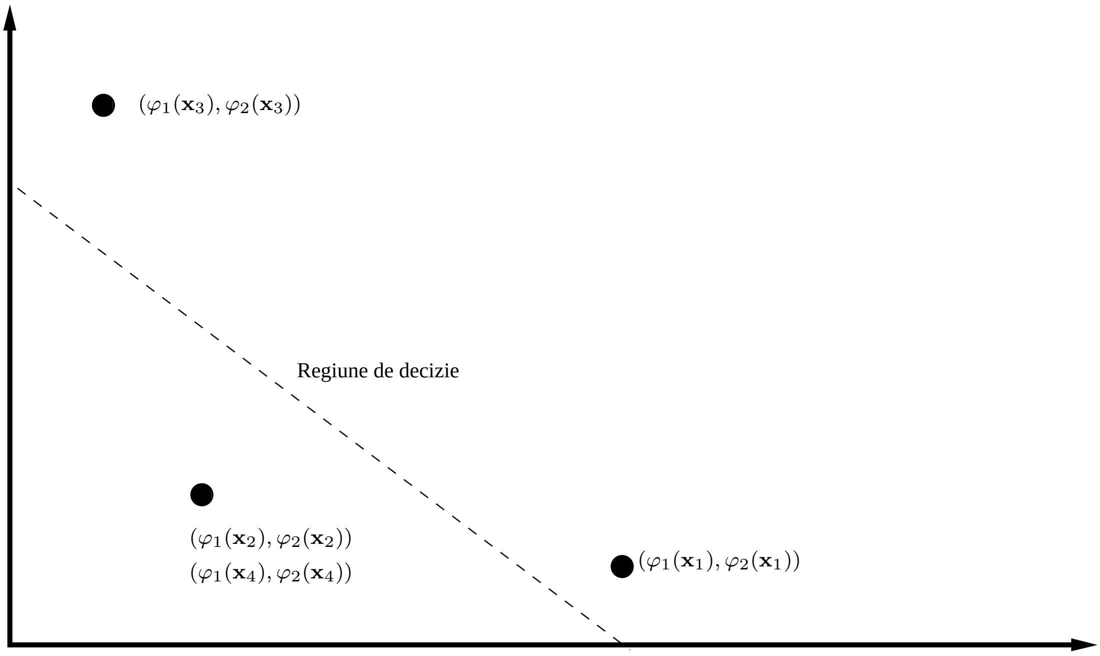
$$\varphi_1(\mathbf{x}) = \exp\left(-\|\mathbf{x} - \mathbf{t}_1\|\right), \mathbf{t}_1 = (1, 1)^t$$

$$\varphi_2(\mathbf{x}) = \exp\left(-\|\mathbf{x} - \mathbf{t}_2\|\right), \mathbf{t}_2 = (0, 0)^t$$

unde $\mathbf{x} \in \mathbb{R}^2$ iar $\|\cdot\|$ este norma Euclidiană L_2 în $\mathbb{R}^2$. Pornind de la un vector de intrare $\mathbf{x} \in \mathbb{R}^2$ se ajunge la un vector tot din $\mathbb{R}^2$ dat de $(\varphi_1(\mathbf{x}), \varphi_2(\mathbf{x}))$. Valorile rezultate pentru funcţiile $\varphi_{1,2}$ calculate în cele 4 puncte ale problemei XOR sunt date în tabelul 8.1. Figura 8.1 dă reprezentarea punctelor tranformate prin aplicarea celor două funcţii. Se observă că problema devine una liniar separabilă, folosind modificări neliniare ale datelor iniţiale; mai mult, nu a fost nevoie în acest caz să se mărească dimensiunea spaţiului de ieşire.

Vector de intrare	$\varphi_1(\mathbf{x}_i)$	$\varphi_2(\mathbf{x}_i)$
$\mathbf{x}_1 = (1,1)^t$	1	0.1353
$\mathbf{x}_2 = (0,1)^t$	0.3678	0.3678
$\mathbf{x}_3 = (0,0)^t$	0.1353	1
$\mathbf{x}_4 = (1,0)^t$	0.3678	0.3678

Tabela 8.1: Valorile funcţiilor φ pentru punctele problemei XOR



8.2 Funcţii cu activare radială

Teorema lui Cover afirmă că pentru o problemă ce nu e liniar separabilă, prin transformare adecvată cresc şansele de a se transforma într–una care e liniar separabilă. Să considerăm o reţea neurală de tip feedforward cu un strat de intrare cu n noduri, un singur strat ascuns şi un strat de ieşire cu un singur nod[1]. Această reţea produce o funcţie de la un spaţiu n–dimensional la unul unidimensional:

$$F : \mathbb{R}^n \to \mathbb{R} \tag{8.2}$$

Funcţia F poate fi văzută ca o hipersuprafaţă $\Gamma \subset \mathbb{R}^{n+1}$; hipersuprafaţa Γ este necunoscută şi se determină pe baza setului de instruire.

Se lucrează în două etape: una de instruire şi alta de generalizare. În

[1]În cele ce urmează în acest capitol, ieşirea unică este pentru simplificarea prezentării. Pentru cazul în care ieşirea este din spaţiul $\mathbb{R}^m$ sau dacă avem o problemă de clasificare cu m clase, stratul de ieşire va avea m neuroni.

etapa de instruire se foloseşte o procedură oarecare prin care se determină hipersuprafaţa Γ, plecând de la setul de date de antrenare, adică se obţine funcţia s. În etapa de generalizare se foloseşte un procedeu de interpolare pentru a determina valori de ieşire corespunzătoare unor vectori din spaţiul de intrare $\mathbb{R}^n$.

Problema de interpolare este:
Dându–se un set de instruire de p perechi

$$\mathcal{S} = \left\{ \left(\mathbf{x}^{(i)}, y^{(i)} \right) | \mathbf{x}^{(i)} \in \mathbb{R}^n, y^{(i)} \in \mathbb{R}, i = \overline{1,p} \right\}$$

se cere să se determine o funcţie $F : \mathbb{R}^n \to \mathbb{R}$ care satisface proprietatea de interpolare:

$$F\left(\mathbf{x}^{(i)}\right) = y^{(i)}, \quad i = \overline{1,p} \tag{8.3}$$

Tehnica funcţiilor cu activare radială (Radial Basis Functions, RBF) consideră că funcţia $F : \mathbb{R}^n \to \mathbb{R}$ are forma:

$$F(\mathbf{x}) = \sum_{i=1}^{p} w_i \varphi_i \left(\left\| \mathbf{x} - \mathbf{x}^{(i)} \right\| \right) \tag{8.4}$$

unde φ_i sunt funcţii neliniare reale, cunoscute ca funcţii cu activare radială. Punctele $\mathbf{x}^{(i)}$ sunt "centrele" (parametri ai) funcţiilor RBF. Pentru un vector $\mathbf{x} \in \mathbb{R}^n$, valoarea returnată de funcţia φ_i se bazează pe distanţa dintre $\mathbf{x}$ şi $\mathbf{x}^{(i)}$.

Impunând condiţia (8.3) asupra formei (8.4), avem următorul sistem liniar în care necunoscutele sunt $w_i, i = \overline{1,p}$:

$$\begin{bmatrix} \varphi_{11} & \varphi_{12} & \cdots & \varphi_{1p} \\ \varphi_{21} & \varphi_{22} & \cdots & \varphi_{2p} \\ \vdots & \vdots & \vdots & \vdots \\ \varphi_{p1} & \varphi_{p2} & \cdots & \varphi_{pp} \end{bmatrix} \cdot \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_p \end{bmatrix} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(p)} \end{bmatrix} \tag{8.5}$$

unde

$$\varphi_{ij} = \varphi_i \left(\left\| \mathbf{x}^{(j)} - \mathbf{x}^{(i)} \right\| \right), \; i, j = \overline{1,p} \tag{8.6}$$

Notăm cu $\mathbf{y} = \left(y^{(1)}, y^{(2)}, \ldots, y^{(p)} \right)^t$, $\mathbf{w} = (w_1, w_2, \ldots, w_p)^t$, $\mathbf{\Phi} = (\varphi_{ij})_{i,j=\overline{1,p}}$. Numim $\mathbf{\Phi}$ matricea de interpolare. Se poate rescrie (8.5) sub forma:

$$\mathbf{\Phi} \cdot \mathbf{w} = \mathbf{y} \tag{8.7}$$

Dacă matricea $\mathbf{\Phi}$ este nesingulară, atunci ponderile sunt $\mathbf{w} = \mathbf{\Phi}^{-1}\mathbf{y}$. Pentru discuţia asupra caracterului nesingular al matricei $\mathbf{\Phi}$, considerăm teorema lui Michelli:

Teorema 4 (Michelli, 1986)**.** *Fie* $\{\mathbf{x}_i\}_{i=\overline{1,p}}$ *un set de puncte distincte din* $\mathbb{R}^n$*. Atunci matricea de interpolare* $\mathbf{\Phi}$ *este inversabilă dacă funcţiile* φ_i *au una din formele:*

1. *funcţie multipătratică:*

$$\varphi_i(r_i) = \sqrt{r_i^2 + c^2},\ c > 0 \tag{8.8}$$

2. *funcţie inversă de multipătratică:*

$$\varphi_i(r_i) = \frac{1}{\sqrt{r_i^2 + c^2}},\ c > 0 \tag{8.9}$$

3. *funcţie Gaussiană:*

$$\varphi_i(r_i) = \exp\left(-\frac{r_i^2}{2\sigma^2}\right),\ \sigma > 0 \tag{8.10}$$

unde în toate cele trei cazuri r_i *este distanţa Euclidiană dintre vectorii* $\mathbf{x}$ *şi* $\mathbf{x}^{(i)}$ *— echivalent: norma diferenţei dintre* $\mathbf{x}$ *şi* $\mathbf{x}^{(i)}$*,* $\left\|\mathbf{x} - \mathbf{x}^{(i)}\right\|$*.*

8.3 Reţele cu funcţii cu activare radială

O reţea cu funcţii cu activare radială este ilustrată în figura 8.1; ea constă din trei straturi:

1. *stratul de intrare*, care constă din n noduri, unde n este dimensiunea spaţiului de intrare.
2. *stratul ascuns*, care e format din acelaşi număr de neuroni ca numărul de date din setul de antrenare, p; fiecare neuron i, $i = \overline{1,p}$ are funcţie cu activare radială $\varphi_i\left(\left\|\mathbf{x} - \mathbf{x}^{(i)}\right\|\right)$;
3. *stratul de ieşire*, care în cazul exemplificat este format dintr-un singur neuron. Stratul de ieşire poate avea m neuroni, pentru a trata problemele de clasificare sau de estimare de probabilitate condiţionată pentru m clase, sau pentru probleme de regresie din $\mathbb{R}^n$ în $\mathbb{R}^m$.

Pentru funcţiile φ_i vom considera în continuare funcţiile Gaussiene:

$$\varphi_i\left(\left\|\mathbf{x} - \mathbf{x}^{(i)}\right\|\right) = \exp\left(-\frac{1}{2\sigma_i^2}\left\|\mathbf{x} - \mathbf{x}^{(i)}\right\|^2\right),\quad i = \overline{1,p} \tag{8.11}$$

unde σ_i este lăţimea unei funcţii Gaussiene centrate în $\mathbf{x}^{(i)}$. De regulă tuturor Gaussienelor li se asigneaza aceeaşi lăţime σ; diferenţa dintre funcţii este dată în acest caz de centrele $\mathbf{x}^{(i)}$.

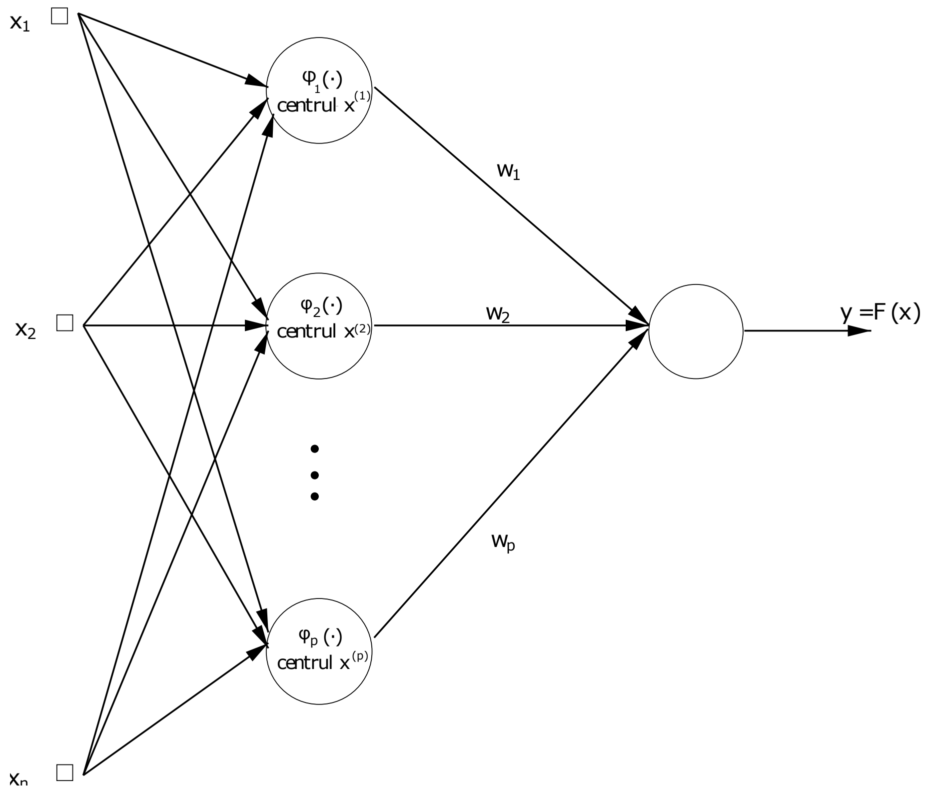


Figura 8.1: Structura unei reţele RBF, plecând de la funcţia de interpolare din ecuaţia 8.4.

Din punct de vedere practic se evită folosirea tuturor datelor din setul de instruire pentru crearea de funcţii de activare de tip radial. Un motiv ar fi că setul $\left\{\left(\mathbf{x}^{(i)}, y^{(i)}\right) | i = \overline{1,p}\right\}$ poate prezenta zgomot, de exemplu datorită erorilor de măsurare. Folosirea unui procedeu de aproximare plecând de la un set de date cu zgomot duce la model de predicţie slab. În plus, numărul de noduri rezultat în reţeaua din figura 8.1 s-ar putea să fie prohibitiv. Ca atare, în practică numărul de noduri din stratul ascuns este mult redus. Funcţia F se transformă într-o funcţie de aproximare de forma:

$$F(\mathbf{x}) = \sum_{i=1}^{k} w_i \varphi_i \left(\|\mathbf{x} - \hat{\boldsymbol{\mu}}_i\|\right) \tag{8.12}$$

unde dimensiunea vectorului de intrare $\mathbf{x}$ este aceeaşi ca şi cea de până acum, $k < p$ iar punctele $\hat{\boldsymbol{\mu}}_i$ nu sunt neapărat din setul de instruire – ele pot proveni dintr–un proces de grupare automată (clustering). Interpretarea ca reţea neurală este dată în figura 8.2. Diferenţele faţă de figura 8.1 sunt că stratul ascuns are k neuroni în loc de p, iar funcţiile de activare radială sunt centrate în vectori $\hat{\boldsymbol{\mu}}_k$ în locul datelor din setul de instruire, $\mathbf{x}^{(i)}$.

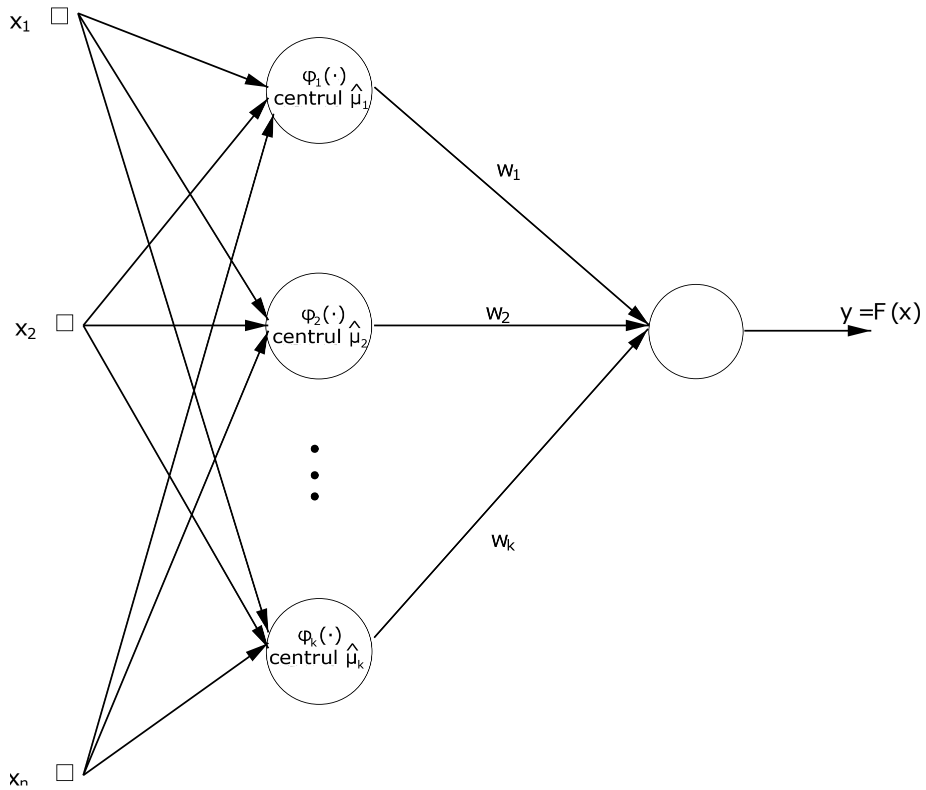


Figura 8.2: Structura unei reţele RBF, folosind mai puţine noduri decât în figura 8.1. Centrii $\hat{\boldsymbol{\mu}}_i$, $i = \overline{1, k}$ se obţin printr-un procedeu de clustering.

Pentru determinarea celor k centri $\hat{\boldsymbol{\mu}}_i$, $1 \leq i \leq k$ ale centrilor din stratul ascuns se poate utiliza o metodă oarecare de grupare automată pe baza similarităţilor (clustering), dar care să producă noţiunea de centroid[2]. Vom prezenta în cele ce urmează metoda K-means clustering.

8.4 Clustering folosind algoritmul K-means

Clustering-ul este o formă de învăţare nesupervizată în care un set de vectori este partiţionat în grupuri. Se urmăreşte minimizarea unei funcţii de cost definită convenabil, care cuantifică disimilaritatea totală a vectorilor. Clusterele ar trebui obţinute de aşa manieră încât vectorii similari să fie grupaţi în acelaşi cluster, iar doi vectori nesimilari să fie dispuşi în clustere diferite.

Considerăm un set de p puncte, $\left\{\mathbf{x}^{(i)}\right\}_{i=\overline{1,p}}$ ce urmează să fie partiţionat

[2]Excludem deci algoritmi de clustering precum DBSCAN sau OPTICS.

în k clustere[3]; de regulă, $k \ll p$. Fie $C(i)$ indicele de cluster de care aparţine vectorul $\mathbf{x}^{(i)}$, $i = \overline{1,p}$. Evident, $1 \leq C(i) \leq k$. Considerăm $d\left(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}\right)$ o măsură a deosebirii – a ne-similarităţii – dintre perechile de vectori $\mathbf{x}^{(i)}$ şi $\mathbf{x}^{(j)}$. Pentru clustering se cere minimizarea funcţiei:

$$J(C) = \frac{1}{2}\sum_{l=1}^{k}\sum_{i:C(i)=l}\sum_{j:C(j)=l} d\left(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}\right) \tag{8.13}$$

unde C este partiţionarea dată de cele k clustere:

$$C = \left\{\{i|1 \leq i \leq p, C(i) = l\}|l = \overline{1,k}\right\}$$

În algoritmul K-means drept măsură de ne–similaritate se foloseşte pătratul distanţei Euclidiene:

$$d\left(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}\right) = \left\|\mathbf{x}^{(i)} - \mathbf{x}^{(j)}\right\|^2 \tag{8.14}$$

În urma procesului de clustering vor rezulta k centroizi – centri de clustere – notaţi $\hat{\boldsymbol{\mu}}_l \in \mathbb{R}^n$, $l = \overline{1,k}$.

Modificăm forma funcţiei de eroare J astfel încât să se ia în considerare distanţele dintre vectorii $\mathbf{x}^{(i)}$ şi centroizii $\hat{\boldsymbol{\mu}}_l$ ai clusterelor de care aparţin:

$$J(C) = \frac{1}{2}\sum_{l=1}^{k}\sum_{i:C(i)=l} \left\|\mathbf{x}^{(i)} - \hat{\boldsymbol{\mu}}_l\right\|^2 \tag{8.15}$$

Presupunând partiţionarea C cunoscută, cum anume se poziţionează centroizii astfel încât să se minimizeze $J(C)$? Algoritmul K-means determină nişte valori pentru $\hat{\boldsymbol{\mu}}_l$ printr-un proces iterativ, astfel încât $J(C)$ să scadă. Este un algoritm euristic, nu garantează faptul că se ajunge la minimul global al lui $J(C)$.

Algoritmul alege aleator k centroizi iniţiali $\hat{\boldsymbol{\mu}}_l^{(1)}$, $l = \overline{1,k}$, iniţializându–i cu valori fie din setul de instruire, sau setate la întâmplare cu valori din spaţiul de intrare, sau conform algoritmului K–means++, a se vedea mai jos. Avem apoi o succesiune de iteraţii cu paşii:

- *Pasul de asignare:* Calculează pentru orice l, $l = 1 \dots k$:

  $$S_l^{(t)} = \left\{\mathbf{x}^{(i)} : \left\|\mathbf{x}^{(i)} - \hat{\boldsymbol{\mu}}_l^{(t)}\right\| \leq \left\|\mathbf{x}^{(i)} - \hat{\boldsymbol{\mu}}_j^{(t)}\right\|, j = \overline{1,k}, j \neq l, i = \overline{1,p}\right\}$$

  adică pentru fiecare punct $\mathbf{x}^{(i)}$ se determină care este cel mai apropiat centroid de care aparţine; $S_l^{(t)}$ este mulţimea vectorilor din setul de instruire ce sunt cel mai apropiate de centroidul $\hat{\boldsymbol{\mu}}_l^{(t)}$, la iteraţia t.

[3] Cel mai frecvent, valoarea lui k este furnizată de utilizator.

- *Modificarea centroizilor:*

$$\hat{\boldsymbol{\mu}}_l^{(t+1)} = \frac{1}{\left|S_l^{(t)}\right|} \cdot \sum_{\mathbf{x}^{(i)} \in S_l^{(t)}} \mathbf{x}^{(i)},\ l = \overline{1,k}$$

  unde $\left|S_l^{(t)}\right|$ este numărul de elemente ale mulţimii $S_l^{(t)}$.

Algoritmul K-means se opreşte atunci când pasul de asignare nu mai modifică mulţimile $S_l^{(t)}$.

În general, algoritmul nu converge către minimul global al funcţiei J; fiind însă rapid în practică – adică necesitând puţini paşi până la oprire – se poate reporni cu alte valori ale centroizilor iniţiali $\hat{\boldsymbol{\mu}}_l^{(1)}$, $l = \overline{1,k}$. Situaţia (centroizii) pentru care $J(C)$ are valoarea cea mai mică în aceste încercări este reţinută.

Se poate consideră că iniţializarea centroizilor iniţiali $\hat{\boldsymbol{\mu}}_l^{(1)}$, $i = \overline{1,k}$ nu ar trebui lăsată la voia întâmplării şi că se poate îmbunătăţi considerabil performanţa algoritmului printr–o alegere îngrijită a lor. Un caz nefavorabil este dat în figura 8.4. Să considerăm un dreptunghi cu laturile de lungime $L \gg l$, având în cele patru vârfuri câte un punct $\mathbf{x}^{(i)} \in \mathbb{R}^2$, $i = \overline{1,4}$. Dacă considerăm $k = 2$ şi centroizii sunt aleşi iniţial la jumătatea laturilor de lungime mai mare, atunci algoritmul se opreşte după o iteraţie cu $J(C) = \frac{1}{2} \cdot 4 \left(\frac{L}{2}\right)^2 = \frac{L^2}{2}$ (punctele $\mathbf{x}^{(1)}$ şi $\mathbf{x}^{(3)}$ aparţin clusterului de centroid $\hat{\boldsymbol{\mu}}_1$, iar celelalte două celui de al doilea cluster). Dacă alegerea punctelor se face ca în figura 8.4, atunci se obţine valoarea $J(C) = \frac{l^2}{2}$ – şi se poate arăta că aceasta este şi valoarea minimă a lui J. Având în vedere că L poate fi luat oricât de mare faţă de l, rezultă că o alegere neinspirată a centroizilor poate să ducă la o valoare oricât de depărtată faţă de optim pentru funcţia J.

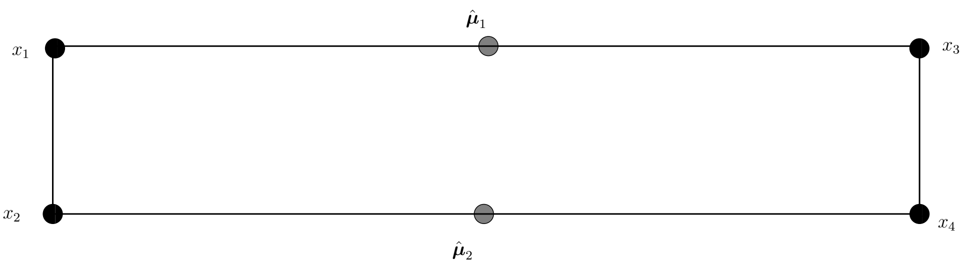


Figura 8.3: Caz nefavorabil pentru K-means la alegerea centroizilor iniţiali.

Ca atare, s–a dezvoltat algoritmul K-means++ care are ca scop determinarea unor centroizi iniţiali aleşi mai potrivit. Alegerea celor K centroizi se face după următorii paşi [12]:

1. Alege primul centroid aleator din setul de antrenare;

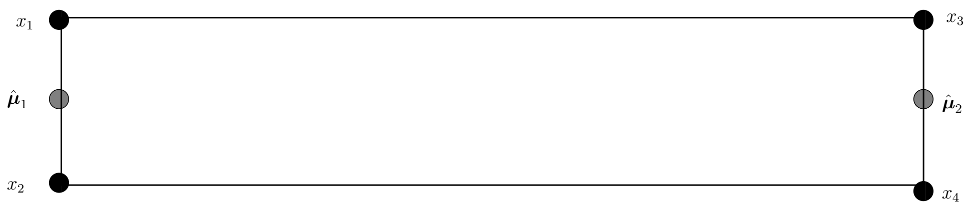


Figura 8.4: Alegerea optimă a centroizilor iniţiali.

2. Pentru fiecare punct ce nu a fost încă ales drept centroid $\mathbf{x}^{(i)}$ calculează $D\left(\mathbf{x}^{(i)}\right)$ $(1 \leq i \leq p)$, distanţa de la el până la cel mai apropiat din centroizii determinaţi până la pasul curent;
3. Alege aleator un nou punct din setul de antrenare, folosind o probabilitate de alegere o funcţie crescătoare cu distanţa dată de D;
4. Repetă paşii 2 şi 3 până când s–au ales toţi cei k centroizi;

Se aplică apoi algoritmul K-means pentru centroizii astfel determinaţi.

Costul suplimentar indus de determinarea celor k centroizi ca mai sus este neglijabil faţă de efectele benefice asupra rezultatului final. Motivaţiile teoretice pentru K-means++ se găsesc în [12].

8.5 Determinarea ponderilor pentru RBF

Determinarea ponderilor legăturilor dintre stratul ascuns şi cel de ieşire este următorul pas. Problema este una de determinare a ponderilor pentru o problemă de regresie liniară şi se tratează cu tehnicile din secţiunile 2.3 şi 2.4.

Pentru cazul în care problema este una de regresie în care vectorii de ieşire sunt din $\mathbb{R}^m$, fiecare neuron de ieşire îşi poate ajusta setul de ponderi independent de ponderile celorlalte ieşiri; se aplică una din cele două metode de mai sus.

8.6 Algoritmul de instruire a reţelei RBF

Sintetizăm pe baza expunerii de până acum procedura de instruire a unei reţele RBF. Stratul de intrare este fix, având numărul de noduri dat de dimensiunea intrării. Stratul ascuns se obţine rulând algoritm de clustering (*e.g.* K-means precedat de K-means++) peste setul de antrenare şi rezultând k centroizi $\hat{\boldsymbol{\mu}}_j, j = \overline{1, k}$. Aceşti centri de clustere devin centrii unor funcţii Gaussiene asignate nodurilor ascunse. Pentru fiecare astfel de

Gaussiană este asignată o aceeaşi lăţime σ, calculată ca:

$$\sigma = \frac{d_{max}}{\sqrt{2 \cdot k}} \tag{8.16}$$

unde d_{max} este distanţa maximă dintre perechile de centroizi. În stratul de ieşire sunt tot atâtea noduri cât este dimensiunea spaţiului de ieşire. Ponderile legăturilor dintre stratul ascuns şi stratul de ieşire se calculează ca pentru o problemă de regresie sau clasificare, definind o funcţie de eroare şi minimizând–o printr-un procedeu adecvat (gradient descent, metoda pseudoinversei etc.)

Capitolul 9

Fuzzy ARTMAP

Reţelele Fuzzy ARTMAP folosesc instruirea supervizată pentru a crea clasificatoare, estimatoare de probabilitate condiţionată şi modele de regresie.

Reţelele Fuzzy ARTMAP (FAM) au capacitatea de învăţare incrementală, posedă o mare parte din proprietăţile dorite pentru sisteme instruibile şi rezolvă dilema stabilitate–plasticitate.

9.1 Învăţarea incrementală

Învăţarea incrementală este o caracteristică asociată unor sisteme adaptive care:

1. agregă cunoştinţe noi din date noi;
2. nu cer acces la datele utilizate pentru a antrena sistemul până la momentul curent;
3. păstrează cunoştinţele deprinse anterior;
4. se pot acomoda cu noi categorii care pot fi introduse de noi date de instruire.

9.2 Proprietăţi dezirabile ale sistemelor instruibile

Pentru un sistem instruibil următoarele proprietăţi sunt văzute ca fiind esenţiale:

1. învăţare rapidă;
2. învăţare din noi date fără a fi nevoie să se reantreneze cu datele parcurse anterior – regăsită în învăţarea incrementală;

3. rezolvarea de probleme neseparabile liniar – o varietate liniară nu este întotdeauna o suprafaţă de separare bună;

4. în cazul unui clasificator: abilitate de a da nu doar clasa de apartenenţă a unui vector de intrare, ci şi plauzibilitatea acestei apartenenţe; sunt favorizaţi aici estimatorii de probabilitate condiţionată de forma $P(clasa|intrare)$; de exemplu, $P(email = spam|continut\ email)$;

5. pentru clasificatori: oferire de explicaţii asupra modului în care datele sunt clasificate, de ce sunt clasificate într–un anume mod; prin această trăsătură se evită tratarea clasificatorului ca o cutie neagră ce nu poate să explice modul de producere a deciziilor;

6. posibilitate de reglare automată a hiperparametrilor modelului; de exemplu, pentru perceptronul multistrat hiperparametrii de interes sunt rata de învăţare, numărul de straturi ascunse, numărul neuronilor din fiecare strat ascuns etc.;

7. aproximarea de funcţii fără a cunoaşte distribuţia iniţială a datelor; rareori datele se supun unei distribuţii clasice;

8. pentru clase care prezintă suprapuneri, să se creeze regiuni în spaţiul de intrare care să realizeze cea mai mică suprapunere; problema asocierilor de tip un vector de intrare–la–mai multe clase trebuie tratată explicit.

9.3 Dilema stabilitate–plasticitate

Un sistem instruibil ar trebui să aibă două proprietăţi:

1. *plasticitate* - înseamnă adaptarea la mediul din care provin vectorii de instruire. Altfel zis, plasticitatea este capacitatea de învăţare.

2. *stabilitate* - se referă la păstrarea cunoştinţelor învăţate anterior.

Atunci când se prezintă noi intrări unei reţele neurale, cele vechi pot fi uitate. Ponderile reţelei trebuie să fie suficient de flexibile pentru a învăţa noi cunoştinţe (trăsătura de plasticitate), dar nu atât de mult încât să uite ceea ce s–a învăţat anterior (trăsătura de stabilitate). Acest conflict dintre stabilitate şi plasticitate se numeşte *dilema stabilitate–plasticitate*. Cele mai multe dintre reţelele neurale existente sunt fie stabile dar incapabile de a învăţa rapid noi vectori, fie plastice dar instabile; de aceea, dilema menţionată este una din problemele de interes în domeniul modelelor instruibile. S-a formulat întrebarea: cum poate un sistem de învăţare să fie atât stabil cât şi plastic?

Dilema a fost abordată de Carpenter şi Grossberg în [?]. Teoria rezonanţei adaptive (Adaptive Resonance Theory, ART) dezvoltată de cei doi autori este unul din răspunsurile concrete date dilemei. De asemenea, sistemul prezintă abilitate de învăţare incrementală şi mare parte din proprietăţile dezirabile ale sistemelor instruibile.

9.4 Fuzzy ARTMAP

Familia Fuzzy ARTMAP de reţele neurale (FAM) este cunoscută ca una din puţinele care posedă capacitate de învăţare incrementală, rezolvă dilema stabilitate-plasticitate şi are multe din proprietăţile dorite pentru un sistem instruibil.

Carpenter si Grossberg au fost interesaţi de obţinerea de sisteme care se pot organiza singure. Paradigma ART poate fi descrisă ca un tip de grupare incrementală a datelor, având posibilitatea de a învăţa fără antrenare supervizată şi este de asemenea în acord cu modelele cognitive şi de comportament. Foloseşte învăţare nesupervizată; reţeaua este capabilă să găsească automat categoria asociată intrării curente sau să creeze una nouă atunci când este nevoie: numărul de neuroni din reţea nu este fixat aprioric.

Reţelele neurale Fuzzy ART sunt capabile să producă rapid o învăţare stabilă a unor categorii de semnale ca răspuns la secvenţe arbitrare de intrări binare sau continue. Fuzzy ART încorporează operatori din teoria mulţimilor fuzzy.

Sistemele de tip Fuzzy ARTMAP învaţă în mod autonom arbitrar de mulţi vectori, formând categorii de recunoaştere în funcţie de succesul de predicţie. Acest sistem de învăţare supervizată este construit dintr-o pereche de module ART capabile de auto-organizare şi obţinere de categorii de recunoaştere stabile.

Succesul reţelelor bazate pe teoria rezonanţei adaptive este dat de avantajele pe care le au faţă de alte reţele multistrat dezvoltate anterior:

- permit crearea dinamică a nodurilor (neuronilor) fără distrugerea celor existente;
- necesită mai puţine cicluri de antrenare cerute, se pot folosi chiar cu învăţare incrementală, adică să treacă o singură dată peste setul de instruire;
- are convergenţă garantată datorită utilizării unor ponderi mărginite şi monotone.

Fuzzy ARTMAP este utilizabil pentru probleme de clasificare, estimare de probabilitate şi regresie. S–a demonstrat că FAM este aproximator universal. Deoarece atât clasificarea cât şi estimarea de probabilitate sunt cazuri particulare ale regresiei, rezultă că FAM poate fi utilizat în orice problemă ce presupune stabilirea de legături dintre două submulţimi din $\mathbb{R}^n$ şi respectiv din $\mathbb{R}^m$.

În final, mai precizăm că FAM mai are o calitate: reprezentarea vectorilor învăţaţi prin categorii facilitează extragerea de reguli sub forma de reguli, aspect esenţial în domeniul extragerii de cunoştinţe din date.

9.4.1 Arhitectura reţelei FAM

O reţea FAM constă într–o pereche de module ART notate ART_a şi ART_b, conectate printr-un modul numit Mapfield, notat F^{ab}. ART_a şi ART_b sunt folosite pentru codificarea vectorilor de intrare şi respectiv de ieşire, iar Mapfield permite asocierea între intrări şi ieşiri. Figura 9.1 conţine componentele unei arhitecturi FAM.

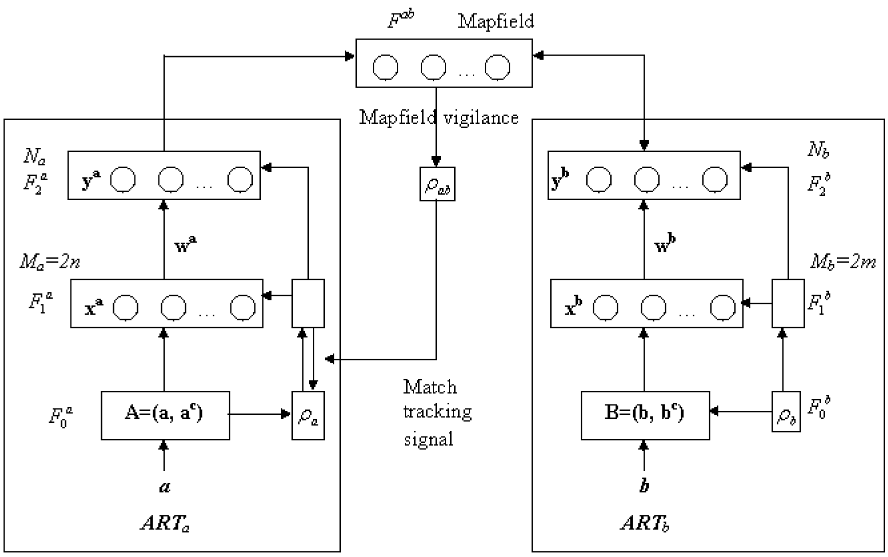


Figura 9.1: Arhitectura Fuzzy ARTMAP

Modulul Fuzzy ART_a conţine stratul de intrare F_1^a şi stratul competitiv F_2^a. Se adaugă de asemenea un strat de preprocesare F_0^a înaintea lui F_1^a. Straturi echivalente apar în ART_b.

Vectorii de intrare iniţiali[1] sunt daţi sub forma:

$$\mathbf{a}^{(i)} = \left(a_1^{(i)}, \ldots, a_n^{(i)}\right),\ a_j^{(i)} \in [0,1],\quad j=\overline{1,n}, i=\overline{1,p} \tag{9.1}$$

p fiind numărul de vectori din setul de instruire.

[1] În acest capitol vectorii sunt văzuţi ca matrice cu o linie.

În cazul în care datele iniţiale nu sunt din intervalul $[0, 1]$, se poate aplica o scalare globală:

$$a_j^{(i)} \to \frac{a_j^{(i)} - MIN}{MAX - MIN}, j = \overline{1, n}, i = \overline{1, p}$$

pentru fiecare vector de intrare $\mathbf{a}^{(i)} = \left(a_1^{(i)}, \ldots, a_n^{(i)}\right)$, $1 \leq i \leq p$, unde MIN şi MAX sunt valoarea minimă şi respectiv maximă din vectorii de intrare:

$$MIN(MAX) = \min(\max) \quad \left\{a_j^{(i)}\right\}, \quad j = \overline{1, n}, i = \overline{1, p}$$

sau se poate lua un minorant (respectiv majorant) al valorilor de intrare. Alternativ, fiecare din trăsături (coloane) poate fi scalată independent de celelalte. În urma aplicării oricărei din aceste scalări, vectorii din setul de instruire vor avea valori în intervalul $[0, 1]$.

O tehnică de preprocesare numită *codificare complementară* este efectuată în cele două module fuzzy ART de către stratul F_0^a, respectiv F_0^b pentru a evita proliferarea nodurilor. S–a dovedit că fără codificarea complementară se vor produce numeroase categorii grupate lângă origine, fără a crea altele care să le înlocuiască. Codificarea complementară este utilizată pentru a obţine vectori normalizaţi, adică vectori cu normă constantă:

$$|\mathbf{A}| = const \tag{9.2}$$

unde $|\cdot|$ este o funcţie normă. În cazul de faţă $|\cdot|$ reprezintă norma L_1: pentru un vector $\mathbf{z} = (z_1, \ldots, z_k)$

$$|\mathbf{z}| = L_1(\mathbf{z}) = \sum_{i=1}^{k} |z_i| \tag{9.3}$$

Fiecare vector de intrare $\mathbf{a} = (a_1, \ldots, a_n)$ produce vectorul:

$$\mathbf{A} = (\mathbf{a}, \mathbf{a}^c) = (\mathbf{a}, \mathbf{1} - \mathbf{a}) = (a_1, \ldots, a_n, 1 - a_1, \ldots, 1 - a_n) \tag{9.4}$$

a cărui normă este:

$$|\mathbf{A}| = \sum_{i=1}^{n} a_i + \sum_{i=1}^{n} (1 - a_i) = n = constant \tag{9.5}$$

motiv pentru care spunem că vectorul $\mathbf{A}$ este normalizat.

Pentru ART_a folosim următoarele notaţii: M_a este numărul de noduri în F_1^a şi N_a este numărul de noduri din F_2^a. Datorită pasului de preprocesare, $M_a = 2n$. Fiecare nod F_2^a reprezintă un grup de intrări similare (numit în alte contexte cluster); vom folosi termenul “categorie” pentru a ne referi la un nod F_2^a. Fiecare categorie F_2^a are propriul set de ponderi adaptive stocate sub forma unui vector:

$$\mathbf{w}_j^a = \left(w_{j,1}^a, \ldots, w_{j,M_a}^a\right), \ j = \overline{1, N_a}. \tag{9.6}$$

Aceste ponderi formează memoria pe termen lung a sistemului. Iniţial, toţi vectorii au valorile:

$$w_{ji}^a = 1,\ j = \overline{1, N_a},\ i = \overline{1, M_a} \tag{9.7}$$

Spunem că un nod din F_2^a este *necomis* dacă nu a învăţat încă nici un vector de intrare, *comis* în caz contrar. Modulul ART_a este responabil cu crearea grupărilor de vectori de intrare. În timpul etapei de învăţare, N_a este numărul de noduri (categorii) comise. Notaţii şi afirmaţii similare se folosesc pentru ART_b, care primeşte vectori m-dimensionali. Pentru o problemă de clasificare, adică o problemă pentru care numărul – total sau iniţial – de clase de ieşire este aprioric cunoscut, indexul de clasă este acelaşi cu indexul de categorie din F_2^b şi astfel ART_b poate fi substituit de un vector N_b-dimensional. Într-o astfel de codificare, valoarea lui N_b poate să crească – dacă noi clase de ieşire sunt adăugate la setul de instruire.

Modulul Mapfield permite FAM să creeze legături între cele două module ART, stabilind legături de tip mulţi–la–unu între categorii din ART_a şi ART_b. Numărul de noduri din F^{ab} este egal cu numărul de noduri din F_2^b. Fiecare nod j din F_2^a este legat cu fiecare nod din F_2^b via un vector de ponderi $\mathbf{w}_j^{ab}$, unde $\mathbf{w}_j^{ab}$ este a j–a linie dintr-o matrice $\mathbf{w}^{ab}$ ($j = \overline{1, N_a}$). Toate ponderile din $\mathbf{w}^{ab}$ sunt iniţializate cu 1:

$$w_{jk}^{ab} = 1,\ j = \overline{1, N_a},\ k = \overline{1, N_b} \tag{9.8}$$

9.4.2 Algoritmul de învăţare pentru FAM

În următorul algoritm, operatorul $\wedge$ este aşa–numitul operator fuzzy AND, care pentru doi vectori

$$\mathbf{p} = (p_1, \ldots, p_k), \mathbf{q} = (q_1, \ldots, q_k) \tag{9.9}$$

cu $0 \leq p_i,\ q_i \leq 1,\ i = \overline{1, k}$ este definit ca:

$$(\mathbf{p} \wedge \mathbf{q})_i = \min(p_i, q_i), i = \overline{1, k} \tag{9.10}$$

1. Se setează parametrul de factor de vigilenţă ρ_a la o valoare egală cu o valoare de bază prestabilită: $\rho_a = \overline{\rho}_a \in [0, 1)$ şi se consideră că toate categoriile din F_2^a sunt neinhibate – adică fiecare nod participă la căutarea unei categorii adecvate pentru vectorul de intrare curent;

2. Pentru fiecare vector de intrare preprocesat $\mathbf{A}$, o funcţie fuzzy este folosită pentru a obţine un răspuns de la fiecare categorie F_2^a:

$$T_j(\mathbf{A}) = \frac{|\mathbf{A} \wedge \mathbf{w}_j^a|}{\alpha_a + |\mathbf{w}_j^a|}, \quad j = \overline{1, N_a} \tag{9.11}$$

3. Fie J indicele de nod neinhibat care dă cea mai mare valoare calculată precum în (9.11):

$$J = \arg\max \{T_j | 1 \leq j \leq N_a \text{ şi nodul } j \text{ nu este inhibat}\} \tag{9.12}$$

4. Verifică condiţia de rezonanţă, i.e. dacă intrarea este suficient de similară cu prototipul câştigătorului:

$$\frac{|\mathbf{A} \wedge \mathbf{w}_J^a|}{|\mathbf{A}|} \geq \rho_a \tag{9.13}$$

   Dacă condiţia este îndeplinită, atunci mergi la pasul 5, altfel inhibă nodul J astfel încât el nu va mai participa la competiţia pentru vectorul curent. Dacă există noduri neinhibate, atunci mergi la pasul 3, altfel recrutează o nouă categorie (creează un nou nod în F_2^a) pentru a reprezenta vectorul de intrare şi fie J indicele acestui nou nod.

5. Un proces similar se desfăşoară şi în ART_b. Fie K indicele nodului câştigător din ART_b. Vectorul de ieşire N_b-dimensional F_2^b este setat la:

$$y_k^b = I(k = K), \quad k = \overline{1, N_b} \tag{9.14}$$

   unde I este funcţia indicator (3.23) de la pagina 42.

   În Mapfield se formează vector de ieşire $\mathbf{x}^{ab}$:

$$\mathbf{x}^{ab} = \mathbf{y}^b \wedge \mathbf{w}_J^{ab} \tag{9.15}$$

6. Un test de verificare în Mapfield controlează potrivirea dintre valoarea prezisă $\mathbf{x}^{ab}$ şi vectorul de ieşire ataşat vectorului de instruire curent $\mathbf{y}^b$:

$$\frac{|\mathbf{x}^{ab}|}{|\mathbf{y}^b|} \geq \rho_{ab} \tag{9.16}$$

   unde $\rho_{ab} \in [0, 1]$ este un parametru de vigilenţă Mapfield. Dacă testul din ecuaţia (9.16) este trecut, atunci se face învăţare ART_a, ART_b şi Mapfield (pasul 7). Altfel, se iniţiază o secvenţă de paşi numită *match tracking* (pasul 8).

7. În modulele fuzzy ART şi în Mapfield se efectuează învăţare:

$$\mathbf{w}_J^{a(new)} = \beta_a \left(\mathbf{A} \wedge \mathbf{w}_J^{a(old)}\right) + (1 - \beta_a)\mathbf{w}_J^{a(old)} \tag{9.17}$$

   (şi analog în ART_b) şi

$$w_{Jk}^{ab} = I(k = K) \tag{9.18}$$

   Se merge la pasul 9.

8. Faza de match tracking, în care se intră doar dacă inecuaţia (9.16) nu e în deplinită: măreşte ρ_a:

$$\rho_a = \frac{|\mathbf{A} \wedge \mathbf{w}_J^a|}{|\mathbf{A}|} + \delta \tag{9.19}$$

unde $0 < \delta < 1$. Dacă $\rho_a > 1$ atunci vectorul curent de instruire este rejectat, altfel mergi la pasul 3.

9. Dacă mai sunt vectori în setul de învăţare, mergi la pasul 1, altfel STOP.

Urmează câteva comentarii privind paşii de mai sus:

1. La pasul 2, fiecare vector este preprocesat datorită straturilor F_0^a şi respectiv F_0^b. $\alpha_a > 0$ este un parametru de alegere. Pentru doi vectori $\mathbf{p}$ şi $\mathbf{q}$, raportul:

$$r = \frac{|\mathbf{p} \wedge \mathbf{q}|}{|\mathbf{q}|} \tag{9.20}$$

cu $0 \leq r \leq 1$ dă gradul în care $\mathbf{p}$ este subset fuzzy al lui $\mathbf{q}$ şi deci pentru $0 < \alpha_a \ll 1$, $T_j(\mathbf{A})$ din ecuaţia 9.11 măsoară gradul în care $\mathbf{A}$ este o submulţime fuzzy a categoriei w_j^a. Dacă se creşte valoarea lui α_a atunci se va mări numărul de categorii, lucru nu întotdeauna benefic. Este deci sugerat ca să se menţină α_a la o valoare mică, de exemplu $\alpha_a = 0.001$; valori mai mici ale lui α_a nu duc la o diferenţă semnificativă.

2. La pasul 3, dacă există mai multe categorii ale căror funcţie de alegere atinge maximul, vom considera pe acea categorie care are indicele minim.

3. Parametrul ρ_a calibrează încrederea minimă pe care ART_a trebuie să o aibă vizavi de o categorie activată de o intrare pentru ca ART_a să accepte această categorie, în loc de a căuta o categorie mai bună. Valori mici ρ_a duc la un grad mare de generalizare şi un număr mai mic de categorii ART_a.

4. Dacă inecuaţia (9.13) este îndeplinită, spunem că avem rezonanţă în ART_a pentru vectorul de intrare curent. Datorită pasului de preprocesare, conform (9.5), numitorul din (9.13) este exact dimensiunea originară a vectorilor de intrare, n.

5. Aceiaşi paşi ca în 1 – 4 sunt efectuaţi în paralel pentru modulul ART_b, dacă nu cumva acesta este substituit cu un vector N_b-dimensional; în acest din urmă caz indicele nodului câştigător K este indicele de clasă corespunzător intrării curente;

6. Când se intră în pasul 5, avem rezonanţă atât în ART_a cât şi în ART_b. Vectorul $\mathbf{x}^{ab}$ dă activarea Mapfield şi se foloseşte atât când ambele module F_2^a şi F_2^b sunt active, *i.e.* la faza de învăţare, cât şi când F_2^a este activ şi F_2^b e inactiv (faza de predicţie). În faza de învăţare $\mathbf{x}^{ab}$ are forma din ecuaţia (9.15); în faza de predicţie acest vector este calculat ca:
$$\mathbf{x}^{ab} = \mathbf{w}_J^{ab} \tag{9.21}$$

7. Atunci când ambele module F_2^a şi F_2^b sunt active – lucru valabil la instruirea reţelei, când se furnizează atât vectorul de intrare cât şi cel de ieşire – a J-a categorie câştigătoare din ART_a va corespunde unui vector de ponderi $\mathbf{w}_J^{ab}$ din Mapfield care leagă nodul F_2^a cu categoria F_2^b prezisă. În paralel, pentru modulul ART_b am obţinut vectorul de ieşire $\mathbf{y}^b$ ca în ecuaţia (9.14). Operaţia fuzzy AND ne asigură că $\mathbf{x}^{ab}$ nu e plin cu valoarea zero dacă şi numai dacă valoarea de ieşire prezisă şi cea actuală coincid. Atunci când categoria J este necomisă avem:
$$\mathbf{w}_J^{ab} = (1, 1, \ldots, 1) \tag{9.22}$$
şi deci $\mathbf{x}^{ab} = \mathbf{y}^b$. Când doar modulul F_2^a este activ, matricea $\mathbf{w}^{ab}$ dă valoarea prezisă: indicele categoriei din ART_b asociată cu intrarea curentă este unica poziţie k din linia j a matricei $\mathbf{w}^{ab}$ pentru care $w_{jk}^{ab} = 1$.

   Ecuaţia (9.18) indică faptul că al J-lea nod din ART_a este legat cu categoria de ieşire K, iar legătura, odată făcută, nu se mai schimbă.

8. Pentru β_a (Pasul 7), există două moduri de învăţare:

   (a) *fast learning* corespunde la a seta $\beta_a = 1$ atât pentru nodurile comise cât şi pentru cele necomise;

   (b) *fast-commit and slow-recode learning* corespunde la a seta $\beta_a = 1$ pentru nod necomis şi $\beta_a < 1$ pentru cele comise.

9. La faza de match tracking, datorită creşterii valorii lui ρ_a conform ecuaţiei (9.19), nodul J nu va mai fi în stare să câştige în competiţiile următoare pentru vectorul de intrare curent. Match tracking–ul va declanşa o nouă căutare în ART_a pentru a găsi un alt nod câştigător. Se poate ca asta să ducă la crearea unui nou nod în ART_a. Căutarea se repetă până când $\rho_a > 1$, sau până când se găseşte o categorie adecvată în ART_a.

10. O valoare mare a lui δ în pasul 8 va creşte numărul de categorii. Se foloseşte de regulă o valoare mică $\delta = 0.001$

11. Ordinea de prezentare a vectorilor din setul de instruire influenţează comportamentul reţelei, adică numărul şi poziţiile categoriilor va diferi. Putem face antrenarea în paralel cu diferite permutări ale setului de intrare, apoi se contorizează voturile pentru fiecare vector care trebuie clasificat.

Pasul de învăţare este repetat până când nu mai este nicio eroare pentru setul de învăţare, sau până se atinge o eroare acceptabilă; dacă se vrea învăţare incrementală atunci se face o singură trecere pe setul de antrenare. După învăţare, FAM poate fi utilizat pentru predicţie. În această fază, stratul F_2^b este inactiv şi F_2^a este activ. Conform ecuaţiei (9.21), predicţia este făcută doar pe baza lui $\mathbf{w}_J^{ab}$.

Există o interpretare geometrică interesantă a categoriilor ART_a: fiecare categorie de intrare $\mathbf{w}_j^a$ se reprezintă ca un hiper–dreptunghi R_j, deoarece vectorul de ponderi poate fi scris ca:

$$\mathbf{w}_j^a = \left(\mathbf{u}_j, \mathbf{v}_j^c\right) \tag{9.23}$$

unde

$$\mathbf{v}_j^c = \left(v_{j1}^c, \dots, v_{jn}^c\right) \tag{9.24}$$

(atât $\mathbf{u}$ cât şi $\mathbf{v}$ au n elemente). Vectorul $\mathbf{u}_j$ defineşte un colţ al hiper–dreptunghiului R_j iar $\mathbf{v}_j$ este colţ diagonal opus lui. Pentru $n = 2$, reprezentarea grafică este dată în figura 9.2. Dimensiunea lui R_j este definită ca:

$$|R_j| = |\mathbf{v}_j - \mathbf{u}_j| \tag{9.25}$$

O interpretare similară este valabilă şi pentru modulul ART_b.

În modulul ART_a, atunci când se creează un nou nod în pasul 4, acesta va fi de fapt vectorul de intrare preprocesat curent: $\mathbf{w}_J^{(new)} = \mathbf{A} = (\mathbf{a}, \mathbf{a}^c)$; cu alte cuvinte, $R_j^{(new)}$ este de fapt un punct reprezentând vectorul de intrare preprocesat A. În timpul fiecărui pas de învăţare fast-learning ($\beta_a = 1$), R_j se expandează la $R_j \oplus \mathbf{a}$, hiper–dreptunghiul minim care conţine R_j şi $\mathbf{a}$ – a se vedea figura 9.3; dacă punctul $\mathbf{A}$ este chiar în interiorul lui R_j, atunci R_j nu se modifică. Colţurile lui $R_j \oplus \mathbf{a}$ sunt definite de $\mathbf{a} \wedge \mathbf{u}_J$ şi $\mathbf{a} \vee \mathbf{v}_J$, unde operatorul $\vee$ este definit ca operatorul fuzzy OR:

$$(\mathbf{p} \vee \mathbf{q})_i = \max\left(p_i, q_i\right),\ i = 1 \dots, n \tag{9.26}$$

pentru $\mathbf{p}$ şi $\mathbf{q}$ vectori precum în ecuaţia (9.9).

Se poate arăta că:

$$|R_j| = n - |\mathbf{w}_j| \tag{9.27}$$

iar hiper–dreptunghiul corespunzător unei categorii nu poate creşte oricât de mult:

$$|R_j| \leq (1 - \rho_a)n \tag{9.28}$$

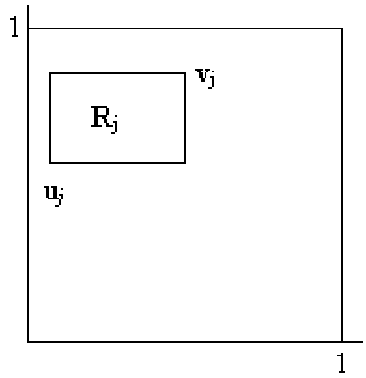


Figura 9.2: Fiecare vector pondere $\mathbf{w}_j^a$ are o interpretare geometrică precum un hiper–dreptunghi R_j cu colţurile definite de $\mathbf{u}_j$ şi $\mathbf{v}_j$

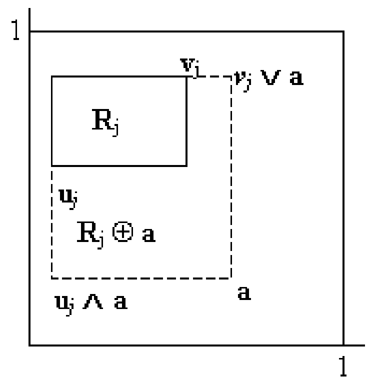


Figura 9.3: Expandarea de categorie în timpul fast learning: de la R_j la hiper–dreptunghiul mai mare conţinând R_j şi $\mathbf{a}$.

Proprietate similară este valabilă şi pentru ART_b.

Aceste proprietăţi sunt sumarizate de teorema:

Teorema 5. *[13] Un sistem Fuzzy ART cu codificarea complementară, fast learning şi termen de vigilenţă constant formează categorii hiper–dreptunghiuri care converg în limită la o secvenţă arbitrară de vectori analogici sau binari. Hiper–dreptunghiurile cresc monoton în toate dimensiunile. Dimensiunea* $|R_j|$ *a unui hiper–dreptunghi este* $n - |\mathbf{w}_j|$*, unde* $\mathbf{w}_j$ *este vectorul pondere corespunzător. Dimensiunea* $|R_j|$ *este mărginită superior de* $n(1 - \rho)$*. Dacă* $0 \leq \rho < 1$*, numărul de categorii este mărginit, chiar dacă numărul de exemplare din setul de antrenare este infinit. Proprietăţi similare au loc pentru fast-learn, slow-recode, exceptând cazul în care este nevoie de prezentări repetate ale fiecărei intrări înainte de stabilizarea sistemului.*

Sumarizând: FAM aplică o învăţare bazată pe potrivire, conform căreia vectorii sunt grupaţi în categorii pe baza unor măsuri de similaritate. Dacă un vector din setul de instruire nu se potriveşte suficient de bine cu o categorie existentă, atunci se va crea una nouă pentru a o reprezenta. Datorită acestui comportament, FAM nu încearcă să minimizeze o funcţie de cost, evitând problemele întâlnite în optimizarea funcţiilor. Strategia de învăţare prezentată este deci diferită de cea bazată pe minimizarea erorii, care cere continuarea antrenării prin epoci suplimentare, dacă valoarea erorii este inacceptabilă sau ponderile nu se stabilizează.

Capitolul 10

Calcul evoluţionist

Calculul evoluţionist este inspirat din teoria evoluţiei dezvoltate de către Charles Darwin şi de genetică – ştiinţa eredităţii, având ca părinte pe Gregor Mendel. Au în comun faptul că folosesc populaţii de elemente care sunt folosite pentru căutarea soluţiei unei probleme, spre deosebire de alte abordări care încercă îmbunătăţirea printr-un proces iterativ a unei singure valori.

10.1 Taxonomie

Calculul evoluţionist se împarte în:

1. algoritmi genetici;
2. programare evoluţionistă;
3. strategii de evoluţie;
4. programare genetică.

Domeniile enumerate au concepte comune; dintre toate, cele mai multe rezultate sunt în domeniului algoritmilor genetici, dar la ora actuală există hibridizări ale acestor 4 arii.

Cel care este creditat ca fiind pionierul domeniului algoritmilor genetici este John T. Holland de la Universitatea din Michigan. El a introdus conceptul de populaţie de indivizi care participă la căutarea unei soluţii; de asemenea, a dat teorema schemelor. El a fost cel care a stabilit operaţiile care trebuie să se aplice unei populaţii genetice - selecţia, încrucişarea şi mutaţia.

Programarea evoluţionistă (avându–l ca pionier pe Larry J. Fogel) foloseşte ca operatori selecţia celui mai potrivit individ şi mutaţia, dar nu şi încrucişarea. În timp ce algoritmii genetici văd procesul evolutiv ca fiind aplicat pe o *populaţie de indivizi din aceeaşi specie*, programarea evoluţionistă vede evoluţia ca aplicându–se unei *populaţii de specii.* Fiecare element din populaţie este interpretat ca o specie întreagă.

Strategiile de evoluţie au fost dezvoltate de Ingo Rechenberg şi Hans-Paul Schwefel, care au experimentat diferite variante de mutaţie pentru rezolvarea unor probleme legate de optimizarea unor suprafeţe aflate în contact cu un fluid. Mutaţiile reprezentau perturbări ale unor stări, efectuând o căutare în vecinătate. Multiplele variante de perturbare au construit un întreg domeniu.

Programarea genetică (Richard Friedberg) a pornit cu coduri program de lungime fixă. Prin modificări efectuate în mod automat asupra acestor programe se dorea obţinerea unor variante de cod optimizate. Esenţiale sunt aici modul de reprezentare a acestor programe şi funcţiile de măsurare a calităţii codului.

De cele mai multe ori, pentru o abordare dintr-unul din cele patru domenii se urmează paşii:

1. Iniţializează populaţia
2. Calculează performanţa fiecărui element din populaţie;
3. Aplică un pas de selecţie;
4. Aplică operaţii precum încrucişarea sau mutaţia;
5. Reia de la pasul 2 până când se îndeplineşte o anumită condiţie.

Diferenţa între domenii constă în detaliile fiecărui pas. Paşii sunt bazaţi pe alegeri de valori aleatoare, ceea ce dă de înţeles că rulări diferite pot duce la valori diferite. Totodată algoritmii nu garantează descoperirea unei valori optime. De cele mai multe ori, însă nu este nevoie să se cunoască exact optimul, ci o valoare suficient de bună. În practică, calculul evoluţionist dă rezultate bune într–un timp rezonabil.

În cele ce urmează vom prezenta algoritmii genetici.

10.2 Algoritmi genetici

Rolul mediului ca factor modelator în teoria evoluţionistă este preluat în algoritmii genetici de către o funcţie scop (sau funcţie obiectiv). Vom detalia algoritmul pentru maximizarea unei funcţii $f : [a, b] \to \mathbb{R}_+^*$. Indivizii care alcătuiesc populaţia se numesc *cromozomi* (şiruri de biţi) şi sunt alcătuiţi din *gene* (biţi).

Constrângerea ca funcţia f să fie strict pozitivă poate fi asigurată prin adunarea unei cantităţi convenabile la funcţia iniţială, dacă are şi porţiuni negative, sau considerarea funcţiei $g(x) = \max(\varepsilon, f(x))$, unde ε etse o constantă strict pozitivă şi mică. Alegerea de a maximiza funcţia obiectiv este convenabilă. Dacă se doreşte minimizarea funcţie, se poate ţine cont de relaţia:

$$\min_x f(x) = -\max_x [-f(x)] \tag{10.1}$$

Se porneşte cu o populaţie iniţială, care este supusă apoi unei secvenţe de procese de tipul:

1. selecţie: indivizii care sunt cei mai buni (considerând valoarea funcţiei f ce se vrea maximizată) sunt favorizaţi să apară de mai multe ori într-o populaţie nouă faţă de indivizii mai puţin performanţi;

2. încrucişare: are loc un schimb de gene între perechi de părinţi, formându-se copii; aceştia se presupune că moştenesc şi combină performanţele părinţilor.

3. mutaţie: se efectuează nişte modificări minore asupra materialului genetic existent.

Pas 1. Crearea unei populaţii iniţiale de cromozomi. Se consideră mai multe valori pentru variabila $x \in [a, b]$. Numărul acestor valori – numit dimensiunea populaţiei – este dat ca parametrul al algoritmului, n, dependent de problemă. Toate valorile sunt cuantificate prin cromozomi care sunt şiruri de k biţi – un bit reprezintă în acest caz o genă a cromozomului, k fiind alt parametru de intrare.

Generarea celor n cromozomi se face aleator, prin setarea fiecărei gene la valoarea 0 sau 1, la întâmplare. Se obţine astfel o populaţie iniţială formată din cromozomii $c_1, \ldots, c_n$.

Fiecare cromozom c (adică şir de k biţi) va produce un număr $x(c)$ din intervalul $[a, b]$, astfel: dacă valoarea în baza 10 a cromozomului este $v(c)$ $(0 \leq v(c) \leq 2^k - 1)$ atunci valoarea asociată din intervalul $[a, b]$ este:

$$x(c) = a + v(c) \cdot \frac{b-a}{2^k - 1} \in \left\{a, a + \frac{b-a}{2^k-1}, a + 2 \cdot \frac{b-a}{2^k-1}, \ldots, b\right\} \tag{10.2}$$

Pas 2. Evoluţia populaţiei. În acest pas se obţin generaţii succesive plecând de la populaţia iniţială; populaţia de la generaţia $g + 1$ se obţine pe baza populaţiei de la generatia g. Operatorii sunt selecţia, împerecherea (crossover, încrucişarea) şi mutaţia.

Pas 2.1. Selecţia. Pentru fiecare cromozom c_i din populatie se calculează funcţia obiectiv $y_i = f(x(c_i))$, $1 \leq i \leq n$. Apoi se însumează valorile funcţiilor obiectiv obţinute pentru fiecare cromozom în parte:

$$S = \sum_{i=1}^{n} y_i \tag{10.3}$$

Pentru fiecare din cei n cromozomi se calculează probabilitatea de selecţie:

$$p_i = \frac{y_i}{S}, 1 \leq i \leq n \tag{10.4}$$

Pentru fiecare cromozom se calculează probabilitatea cumulativă de selecţie:

$$q_j = \sum_{i=1}^{j} p_i, 1 \leq j \leq n \tag{10.5}$$

Remarcăm că se obţine $0 < p_1 = q_1 < q_2 < \cdots < q_n = 1$. Cu cât cromozomul c_i determină o valoare mai mare pentru funcţia f (*i.e.* cu cât valoarea $f(x(c_i))$ este mai mare), cu atât diferenţa dintre q_i şi q_{i-1} este mai mare.

Se selectează n numere aleatoare uniform distribuite în $(0, 1]$. Pentru fiecare număr, dacă el se găseşte în intervalul $(0, q_1]$ atunci cromozomul c_1 este ales şi depus într-o populaţie nouă; dacă acest număr se află în intervalul $(q_i, q_{i+1}]$ atunci se alege cromozomul c_{i+1}. Remarcăm ca numărul de cromozomi prezenţi în noua populaţie este tot n. Cu cât valoarea $y = f(x(c))$ asociată unui cromozom c este mai mare, cu atât cresc şansele lui spre a fi selectat şi depus în noua populaţie. Este foarte probabil ca un astfel de cromozom valoros să apară de mai multe ori in populaţia nouă; de asemenea, este foarte probabil ca un cromozom cu o valoare mică pentru funcţia f să nu apară deloc.

Pas 2.2. Încrucişarea (împerecherea, crossover) Pentru fiecare cromozom care a rezultat la pasul anterior se alege o valoare aleatoare, uniform distribuită în intervalul $(0, 1]$. Dacă această valoare este mai mică decât un parametru p_c (parametru al aplicaţiei, *e.g.* 0.1), atunci cromozomul este ales pentru incrucişare. Se procedează astfel încât să se obţină un număr par de cromozomi (de exemplu se renunţă la ultimul dacă numărul lor este impar).

Cromozomii aleşi se încrucişează astfel: primul selectat cu al doilea selectat, al 3-lea cu al 4-lea etc. Încrucişarea decurge astfel:

- se alege un număr aleator t intre 1 şi $k - 1$;
- se obţin 2 cromozomi copii astfel: primul va conţine primele t gene ale primului părinte şi ultimele $k - t$ gene ale celui de–al doilea părinte; al doilea copil conţine primele t gene ale celui de–al doilea părinte şi ultimele $k - t$ gene ale primului părinte;
- cei doi cromozomi copii îi vor înlocui în populaţie pe părinţii din care provin.

Pas 2.3. Mutaţia. Populaţiei obţinute i se aplică operator de mutaţie, astfel: pentru fiecare genă a fiecărui cromozom se alege o valoare aleatoare, uniform distribuită în $(0, 1]$; dacă acest număr este mai mic decât o probabilitate de mutaţie p_m (parametru al aplicaţiei, *e.g.* $p_m = 0.01$), atunci se modifică valoarea curentă a genei cu complementul său faţă de 1.

Populaţia obţinută în pasul 2 reia ciclul de evoluţie. După ce se obţin câteva astfel de generaţii (sau se epuizează un timp alocat procesului, sau se observă că media populaţiei nu se îmbunătăţeşte în ultimele iteraţii efectuate), se raportează valoarea celui mai bun cromozom din ultima generaţie[1].

Avantajul primar al algoritmilor genetici constă în schimbul de informaţie dintre indivizi realizat la etapa de încrucişare, adică schimbarea de blocuri de date care au evoluat. O utilizare eficientă a algoritmilor genetici presupune crearea unor structuri de date pentru gene şi a unor operatori adecvaţi problemei ce trebuie rezolvată[2] – a se vedea secţiunea 10.4.

10.3 Fundamente teoretice

Studiul comportamentului algoritmilor genetici se face pe baza unor scheme (sau şabloane) care descriu colecţii de cromozomi. O schemă se reprezintă ca un şir de caractere construit cu simbolurile "0", "1" şi "*", unde "*" poate fi substituit cu orice bit; simbolul "*" poate apărea de oricâte ori, inclusiv niciodată. De exemplu, schema $(*0101)$ se potriveşte cu doi cromozomi[3]: (00101) şi (10101). Dacă o schemă are l simboluri "*", atunci ea poate să fie reprezentată de 2^l cromozomi, iar un cromozom de lungime k poate fi descris de $C_k^0 + C_k^1 + \cdots + C_k^k = 2^k$ scheme.

Pentru o schemă S definim ordinul ei (şi îl notăm cu $o(S)$) numărul de poziţii pe care se află valorile 0 sau 1, adică numărul de poziţii fixate. De exemplu, pentru schema $S = (*\ 0\ *\ 1\ 1\ 0)$, $o(S) = 4$. Ordinul unei scheme dă gradul de specializare a ei şi este utilă mai departe în calcularea probabilităţii de supravieţuire a sa în cadrul mutaţiilor.

Lungimea unei scheme S, notată cu $\delta(S)$, este distanţa dintre prima şi ultima poziţie fixată. Pentru schema dată mai sus, $\delta(S) = 6 - 2 = 4$ (sau $\delta(S) = 5 - 1 = 4$, după cum indicierea începe de la 1 sau de la 0). Noţiunea de lungime a unei scheme este utilă pentru calculul probabilităţii de supravieţuire a unei scheme în cadrul operaţiilor de încrucişare.

Pentru o populaţie de indivizi aflată la momentul t al evoluţiei, vom nota cu $n(S, t)$ numărul de cromozomi din populaţie care reprezintă (se potrivesc cu) schema S. De asemenea, vom considera valoarea medie a schemei din populaţia de la un timp t, notată cu $f(S, t)$ şi definită ca suma valorilor

[1] În practică se preferă strategia elitistă: se returnează cel mai bun individ al tuturor generaţiilor.

[2] S-a stabilit "ecuaţia" Genetic Algorithms + Data Structures = Evolution Programs, [14].

[3] În acest caz spunem că schema este reprezentată de cei doi cromozomi.

cromozomilor din populaţie care reprezintă schema S împărţită la numărul acestor cromozomi, $n(S, t)$.

La pasul de selecţie, un cromozom A este copiat în populaţia următoare cu probabilitatea:

$$P(A) = \frac{f(A)}{\sum\limits_{\text{cromozom } c} f(c)} \tag{10.6}$$

unde însumarea se face după toţi cromozomii c ai populaţiei curente. Reamintind că n este numărul de cromozomi din populaţie, avem:

$$n(S, t+1) = n(S, t) \cdot \frac{f(S, t)}{\frac{1}{n} \sum\limits_{\text{cromozom } c} f(c)} \tag{10.7}$$

Cantitatea $\overline{f(t)} = \sum_{\text{cromozom } c} f(c)/n$ este chiar valoarea medie a populaţiei de la momentul t, deci avem:

$$n(S, t+1) = n(S, t) \cdot \frac{f(S, t)}{\overline{f(t)}} \tag{10.8}$$

Numărul de reprezentanţi ai schemei S care vor exista la momentul $t+1$ este dependent de valoarea schemei dată de cromozomii care există în populaţia de la momentul t. De exemplu, o schemă S care produce o valoare relativ mare a lui $f(S, t)$ faţă de $\overline{f(t)}$ va impune creşterea numărului de reprezentanţi ai săi. Dacă presupunem de exemplu că $f(S, t) = \overline{f(t)} + \varepsilon \cdot \overline{f(t)} = \overline{f(t)}(1 + \varepsilon)$, $\forall t > 0$ (unde $\varepsilon > 0$) atunci se poate arăta prin inducţie că:

$$n(S, t) = n(S, 0)(1 + \varepsilon)^t, \ \forall t \in \{1, 2, \dots\} \tag{10.9}$$

adică pentru scheme care au valoare medie desupra valorii medii a populaţiei numărul de reprezentanţi va creşte exponenţial în timp – respectiv dacă valoarea schemei este sub medie, numărul de reprezentanţi obţinuti prin selecţie scade exponenţial.

În ceea ce priveşte încrucişarea, să presupunem că cromozomul cu 7 gene $c = (1010100)$ este selectat pentru reproducere; există 2^7 scheme care îl au pe c drept reprezentant, de exemplu:

$$S_1 = (*01 * * * *) \tag{10.10}$$

şi

$$S_2 = (1 * * * *0*) \tag{10.11}$$

Să presupunem că în procesul de încrucişare tăietura se face după a patra genă:

$$\begin{array}{rcl} c & = & (1 \;\; 0 \;\; 1 \;\; 0 \;\; | \;\; 1 \;\; 0 \;\; 0) \\ S_1 & = & (* \;\; 0 \;\; 1 \;\; * \;\; | \;\; * \;\; * \;\; *) \\ S_2 & = & (0 \;\; * \;\; * \;\; * \;\; | \;\; * \;\; 0 \;\; *) \end{array} \tag{10.12}$$

Se observă că pentru exemplul considerat schema S_1 sigur se va regăsi într–un descendent (deci schema supravieţuieşte), deoarece valorile 0 şi 1 se regăsesc pe poziţiile iniţiale, în timp ce S_2 are şanse de a fi distrusă[4]. Intuitiv, este clar faptul că lungimea mică a schemei S_1 măreşte şansa de supravieţuire, faţă de S_2 care poate fi uşor "spartă" în cromozomii copii. Desigur, contează poziţia tăieturii.

Tăietura poate să apară uniform aleator (echiprobabil) în $k-1$ poziţii. Probabilitatea de distrugere a unei scheme este:

$$P_d(S) = \frac{\delta(S)}{k-1} \tag{10.13}$$

şi evident probabilitatea evenimentului contrar, reprezentând supravieţuirea schemei este

$$P_s(S) = 1 - P_d(S) = 1 - \frac{\delta(S)}{k-1} \tag{10.14}$$

Conform strategiei de alegere a cromozomilor ce se supun împerecherii, probabilitatea ca un cromozom să participe la încrucişare este p_c, deci probabilitatea de supravieţuire a unei scheme S este:

$$P_s(S) = 1 - p_c \cdot \frac{\delta(S)}{k-1} \tag{10.15}$$

Se mai poate lua în considerare faptul că o schemă S poate totuşi să supravieţuiască, dacă cromozomii care se încrucişează au pe poziţiile fixe ale schemei chiar valorile din S. Aşa ceva este posibil şi trebuie considerat ca mărind şansele de supravieţuire a unei scheme. Ca atare, şansa de supravieţuire este de fapt dată printr–o inegalitate:

$$P_s(S) \geq 1 - p_c \cdot \frac{\delta(S)}{k-1} \tag{10.16}$$

deci schemele de lungime mică au şanse crescute de supravieţuire.

Combinând rezultatele obţinute pentru partea de selecţie şi încrucişare, obţinem:

$$n(S, t+1) \geq n(S, t) \cdot \frac{f(S, t)}{\overline{f(t)}} \cdot \left[1 - p_c \cdot \frac{\delta(S)}{k-1}\right] \tag{10.17}$$

Mutaţia schimbă aleator biţi din cromozom cu complementul lor. Este clar că pentru ca o schemă să supravieţuiască, poziţiile sale fixe nu trebuie să fie alese pentru mutaţie. Probabilitatea ca un bit oarecare să nu fie modificat este $(1-p_m)$. Alegerile biţilor care să sufere mutaţie sunt evenimente independente, deci probabilitatea ca cei $o(S)$ biţi ficşi ai unei scheme să se menţină (şi deci ca întreaga schemă să se menţină) este:

$$P_s(S) = (1 - p_m)^{o(S)} \tag{10.18}$$

[4] S_2 nu e distrusă, însă, dacă al doilea cromozom care participa la încrucişare vine cu aceleaşi gene pe poziţiile fixe din schemă.

Pentru că $p_m \ll 1$, putem aproxima $(1-p_m)^{o(S)}$ cu $1-p_m o(S)$. Am obţinut că schemele cu ordinul mic au şanse crescute de supravieţuire.

Efectul combinat al operaţiilor de selecţie, încrucişare, mutaţie este deci:

$$n(S, t+1) \geq n(S,t) \cdot \frac{f(S,t)}{\overline{f(t)}} \cdot \left[1 - p_c \cdot \frac{\delta(S)}{k-1} - p_m \cdot o(S)\right] \tag{10.19}$$

Se poate da acum enunţul teoremei schemelor, teorema fundamentală a algoritmilor genetici datorată lui Holland (1975):

Teorema 6 (Teorema schemelor)**.** *Schemele scurte, de ordin mic, cu valoare peste medie cresc ca număr de reprezentanţi în decursul generaţiilor.*

S–a formulat următoarea ipoteză:
Ipoteza blocurilor de construcţie, [14]. Un algoritm genetic execută un proces de căutare prin suprapunerea unor scheme scurte, de ordin mic şi de valoare mare, numite blocuri de construcţie. Se poate arăta că pentru o populaţie de n cromozomi, numărul de scheme efectiv procesate este în ordinul lui n^3, ceea ce dă caracter de paralelism implicit al algoritmilor genetici: se procesează nu doar o singură schemă, ci mai multe.

10.4 Problema reprezentării datelor în algoritmii genetici

Reprezentarea indivizilor ca şiruri de biţi este nenaturală pentru multe probleme practice. Să presupunem, de exemplu, problema comis-voiajorului: fiind date n oraşe şi distanţele dintre ele, să se determine un tur al lor, astfel încât fiecare oraş să fie vizitat exact o singură dată, să se revină la oraşul de plecare iar costul total al drumului să fie minim[5]. O soluţie este dată ca o permutare a mulţimii $\{1, \ldots, n\}$.

Pentru cazul $n = 20$, dacă folosim reprezentarea binară, putem vedea că cinci biţi sunt suficienţi pentru a reprezenta orice număr de la 1 la 20, deci ar trebui să folosim $20 \cdot 5 = 100$ de biţi pentru reprezentarea unei soluţii potenţiale. Să presupunem că la un moment dat avem grupul de 5 biţi 01101 reprezentând oraşul cu numărul 13; prin aplicarea mutaţiei este posibil să se ajungă la valoarea binară 11101, adică în zecimal 29, un oraş care nu există. S-ar obţine deci o valoare invalidă datorată unei reprezentări neadecvate a elementelor din problemă sau a unor operatori care nu sunt adaptaţi corespunzător. La fel de bine, se poate ca prin mutaţie sau încrucişare să se obţină valori de oraşe repetate, deci un ciclu prematur.

Pentru cei 100 de biţi asociaţi problemei, spaţiul de căutare realizat este $2^{100} \simeq 10^{30}$, în timp ce mulţimea tuturor ciclurilor hamiltoniene este – considerând primul oraş ca fiind fixat si neconsiderând soluţiile simetrice

[5]În termeni de grafuri: se cere determinarea unui ciclu Hamiltonian de lungime minimă.

de forma $A \to B \to C \to A \equiv A \to C \to B \to A$ – mulţimea permutărilor cu $19!/2 \ < 10^{17}$ elemente. În situaţia dată deducem că utilizarea unei codificări binare conduce la un spaţiu de căutare mărit artificial, existând zone mari din spaţiul binar care nu corespund unor soluţii viabile.

Alte exemple de probleme aflate în aceeaşi situaţie pot fi încă date; se ajunge la concluzia că varianta naivă de reprezentare a valorilor şi a operatorilor genetici nu se potriveşte neapărat la orice problemă de căutare. Modelarea unui individ şi a operatorilor asociaţi trebuie să se facă ţinând cont de domeniu şi de particularităţile problemei. Vor fi exemplificate codificări adecvate pentru câteva probleme clasice.

O altă problemă care trebuie tratată este: cum procedăm când există constrângeri? De exemplu, dacă vrem să maximizăm funcţia:

$$f(x, y) = x^2 - y^3 + 2 \cdot x \cdot \sin(y) \tag{10.20}$$

cu condiţia ca variabilele x şi y să satisfacă constrângerea:

$$1 \leq x^3 - \cos(y) + y^2 \leq 5 \tag{10.21}$$

cum încorporăm restricţia în algoritmul genetic? Dacă folosim varianta clasică de codificare a unui individ, împreună cu operatorii de încrucişare şi de mutaţie prezentaţi, cum asigurăm faptul că operatorii daţi nu duc indivizii în zone în care constrângerea (10.21) nu este îndeplinită?

Pentru această din urmă problemă s–au dat următoarele variante:

1. impunerea de penalizări pentru indivizii care încalcă constrângerile;

2. implementarea unei metode de "reparare" a indivizilor care nu satisfac constrângerile;

3. implementarea unor operatori de încrucişare şi de mutaţie care păstrează indivizii în condiţiile impuse.

Pe marginea fiecăreia din cele trei variante există multiple versiuni:

- pentru penalizări, valoarea acestora poate fi constantă, sau să varieze cu gradul în care se încalcă constrângerile date; această ultimă variantă poate fi codificată sub forma unei funcţii logaritmice, liniare, pătratice etc. O formă extremă de penalizare este eliminarea indivizilor care încalcă restricţiile, dar trebuie dat răspuns la întrebarea: cu ce se umple locul lăsat gol prin eliminare? sau cumva se permite populaţiei de dimensiune variabilă? cei mai mulţi autori afirmă că această eliminare este prea dură, în timp ce menţinerea unor indivizi penalizaţi oferă variabilitate populaţiei – se pot produce descendenţi valizi, chiar şi din cei care nu respectă constrângerile.

- pentru algoritmii de reparare – este posibil să se integreze cunoştinţe din domeniu în metodele de corecţie; trebuie zis însă că elaborarea unui algoritm de corecţie poate uneori să fie o problemă la fel de grea ca şi rezolvarea problemei de la care s–a plecat.

- pentru ultima variantă – este cunoscut deja că orice tip de date trebuie să vină cu un set de operatori dedicaţi care să permită prelucrarea tipurilor; o codificare potrivită a problemei împreună operatorii asociaţi trebuie să favorizeze (ideal: să garanteze) generarea de indivizi valizi. Aici se intervine cu cunoştinţe despre problema care trebuie rezolvată, cunoştinţe care, prin implementare, favorizează obţinerea de indivizi care nu încalcă (prea mult, sau deloc) restricţiile;

Pentru fiecare din abordări s–au studiat variante şi comportamente; studiul s–a făcut în mare măsură empiric, pe probleme concrete; la ora actuală, un rezultat precum teorema schemei este inexistent pentru alte codificări decât cea binară. Desigur, se poate folosi şi o combinaţie a celor trei variante de mai sus.

Pentru numeroase probleme practice s–a constatat experimental că reprezentarea adecvată a indivizilor, împreună cu definirea unor operatori particularizaţi şi cele 3 metode de mai sus dau rezultate mai bune decât aplicarea *ad literam* a algoritmului genetic peste o formă binarizată a problemei.

Vom exemplifica pentru problema discretă a rucsacului: se dă un rucsac de capacitate C, un set de m obiecte având greutăţile $G_i > 0$ şi valorile asociate $V_i > 0$, $1 \leq i \leq m$. Un obiect poate fi luat doar în întregime în rucsac; problema este: care sunt obiectele care trebuie încărcate, astfel încât greutatea totală să nu depăşească C iar valoarea cumulată să fie maximă?

Problema este NP-completă, deci la ora actuală nu cunoaştem un algoritm de complexitate polinomială care să o rezolve. Multe probleme pot fi reduse la aceasta, de aici interesul acordat.

O reprezentare naturală a unui individ – respectiv încărcare de rucsac – este un vector $\mathbf{x}$ cu elementele x_i, $1 \leq i \leq m$, $x_i \in \{0, 1\}$, valoarea 0 însemnând că obiectul nu este luat, iar 1 - că e adăugat în rucsac. Se impune, evident, condiţia de viabilitate a unui vector $\mathbf{x}$:

$$\sum_{i=1}^{m} x_i \cdot G_i \leq C$$

iar funcţia de maximizat – numită şi profit în acest caz – este:

$$P(\mathbf{x}) = \sum_{i=1}^{m} x_i \cdot V_i$$

10.4.1 Varianta cu penalizare

Pentru fiecare individ $\mathbf{x}$ se va considera valoarea sa $val(\mathbf{x})$:

$$val(\mathbf{x}) = \sum_{i=1}^{m} x_i \cdot V_i - Pen(\mathbf{x})$$

unde $Pen(\cdot)$ este funcţia de penalizare:

$$Pen(\mathbf{x}) \begin{cases} = 0, & \text{dacă } \mathbf{x} \text{ este viabil} \\ > 0, & \text{dacă } \mathbf{x} \text{ nu este viabil} \end{cases}$$

Dacă valoarea funcţiei de penalizare depinde de gradul în care se face încălcarea restricţiilor – gradul de încălcare poate fi de exemplu de diferenţa dintre $\sum_{i=1}^{m} x_i \cdot G_i$ şi C – atunci se poate folosi o funcţie de tip logaritmic, liniar, pătratic, exponenţial etc. Efectele alegerii unei asemenea funcţii au fost analizate pe diferite situaţii; a se vedea [14] pentru rezultate experimentale şi interpretarea lor.

10.4.2 Varianta cu reparare

Putem folosi aici tot codificarea binară. Algoritmul de corectare este simplu: dacă setul de obiecte ales depăşeşte ca greutate totală capacitatea C, atunci se scot obiecte până când greutatea celor rămase devine acceptabilă (cel mult G). Vom transforma deci vectorul $\mathbf{x}$ în $\mathbf{x}' = (x'_1, \ldots, x'_m)$ astfel încât $\sum_{i=1}^{m} x'_i G_i \leq C$.

Algoritmul de reparare a unui individ este:

Listing 10.1: Repararea unui vector invalid

```
function reparare(x, G, C) returns a vector
begin
 rucsac-supraincarcat := false
 x' := x
 if ∑_{i=1}^{m} x'_i · G_i > C
  then rucsac-supraincarcat := true
 end if
 while rucsac-supraincarct = true
  i:= selecteaza un obiect din rucsac (#)
  scoate obiectul i din rucsac: x'_i := 0
  if ∑_{i=1}^{m} x'_i · G_i ≤ C
   then rucsac-supraincarcat := false
  end if
 end while
 return x'
end
```

În ce priveşte metoda de selectare a lui i din linia marcată cu (#), avem variantele:

- (reparare aleatoare) valoarea i se alege aleator din setul indicilor obiectelor care se găsesc în rucsac;
- (reparare greedy) se alege obiectul cel mai uşor, sau cel care are raportul P_i/G_i cel mai mic.

10.4.3 Codificarea adecvată a indivizilor

Vom prezenta o strategie de codificare a indivizilor, diferită de cea binară utilizată până acum, numită reprezentarea ordinală. Codificarea este larg utilizată şi în alte probleme care presupun manipularea unor secvenţe de valori, de exemplu în problema comis-voiajorului. Vectorul $\mathbf{x}$ este cu cele m componente în baza 10, fiecare element x_i având proprietatea că $1 \leq x_i \leq m - i + 1$, $\forall i \in \{1, \ldots, m\}$. De exemplu, pentru vectorul $\mathbf{x} = (3, 3, 4, 1, 1, 1)$ asociat listei de obiecte $L = (1, 2, 3, 4, 5, 6)$, decodificarea se obţine astfel: se scoate elementul de indice $x_1 = 3$ din lista L, adică obiectul 3 şi îl adăugăm în rucsac; L devine $(1, 2, 4, 5, 6)$; apoi se scoate elementul de indice $x_2 = 3$ (adică obiectul 4) din L şi îl adăugăm în rucsac, L devine $(1, 2, 5, 6)$; se scoate elementul de indice $x_3 = 4$ din L, adică obiectul 6, L devine $(1, 2, 5)$ etc. Obţinem astfel ordinea de depunere în rucsac: 3, 4, 6, 1, 2, 5. Fiecare cromozom codifică astfel o ordine de adăugare a obiectelor în rucsac. Adăugarea obiectului în rucsac se face numai dacă nu duce la depăşirea capacităţii C.

Se poate vedea că operaţie de încrucişare va duce întotdeauna la copii valizi, adică pentru un copil $\mathbf{z} = (z_1, \ldots, z_m)$ avem că $1 \leq z_i \leq m - i + 1$ dacă şi părinţii au aceeasi proprietate. Mutaţia este similară cu cea de la cazul binar: o componentă aleasă pentru mutaţie, fie ea x_i este modificată cu o valoare aleatoare uniform distribuită în mulţimea $\{1, \ldots, m - i + 1\}$ diferită de x_i.

Listing 10.2: Utilizarea codificării ordinale

```
procedure decodificare(x)
begin
 construieste o lista L de obiecte
 greutateTotala := 0
 profitTotal := 0
 for i = 1,n
  j := x_i
  o := L_j
  sterge elementul al j-lea din lista L
  if greutateTotala + G_o ≤ C
   then begin
```

```
      greutateTotala := greutateTotala + G_o
      profitTotal := profitTotal + P_o
    end
  end if
 end for
end
```

Lista L se poate crea într–o ordine aleatoare, ca o permutare a mulţimii $\{1, \dots, n\}$. Indivizii rezultaţi — adică cei care realizează populaţia iniţială — vor fi deci generaţi aleatori.

Rezultate experimentale pentru cele trei variante de rezolvare sunt date în [14]. Concluziile experimentelor însă nu pot fi generalizate la orice problemă care vine cu impunere de restricţii. Se exemplifică însă că diferenţele de performanţă pot fi mari pentru aceste abordări.

10.5 Exemplu: problema orarului

Problema orarului este o altă situaţia practică care se abordează prin intermediul algoritmilor genetici. Problema este NP-completă. Are diferite enunţuri, vom prezenta varianta dată în [14].

Se dau următoarele:

- o mulţime de profesori $\{T_1, \dots, T_m\}$
- o listă de intervale de timp (ore) $\{H_1, \dots, H_n\}$
- o listă de săli de clasă $\{C_1, \dots, C_k\}$

Orarul *trebuie* să respecte următoarele cerinţe:

- există un număr predefinit de ore pentru fiecare profesor şi clasă;
- la un moment dat, la o clasă predă un singur profesor;
- un profesor nu poate preda la mai multe clase simultan;
- la fiecare clasă programată la o anumită oră trebuie să existe exact un profesor

Mai avem şi constrângeri care *ar trebui* respectate; acestea sunt legate de:

- nicio "fereastră" în orarul elevilor, cât mai puţine în cel al profesorilor;
- preferinţe exprimate de profesori sau elevi: ore doar într–o anumită parte a zilei sau săptămânii;
- împărţire cât mai echilibrată a orelor;

- număr maxim de ore pe zi pentru elevi/profesori

Codificarea binară pentru această problemă, cu toate că este posibilă, poate apărea drept nenaturală; mai mult decât atât, există riscul ca spaţiul de căutare să se mărească artificial, precum la problema comis voiajorului. Putem să codificăm un orar (un individ) ca fiind o matrice **O** cu m linii şi n coloane, unde liniile corespund profesorilor iar coloanele – orelor disponibile. Fiecare celulă este fie liberă, fie conţine o clasă C_i, $1 \leq i \leq k$.

Pentru reprezentarea dată, operatorii genetici ar putea fi[6]:

- mutaţia de ordin k: se iau 2 secvenţe adiacente formate din p elemente şi se interschimbă
- mutaţia de zile: se iau două zile şi se interschimbă între ele
- încrucişare: se porneşte de la două orare părinte O_1 şi O_2, se efectuează tăietură pe orizontală sau pe verticală şi se face interschimbarea de porţiuni, întocmai ca la cromozomii binari

Este posibil ca să fie nevoie să se intervină cu algoritmi de corecţie după aplicarea unor astfel de operatori. Din punct de vedere practic, abordarea prin algoritmi genetici este confirmată ca o metodă funcţională de către mai mulţi autori.

[6] Dar nimic nu ne împiedică să concepem alţi operatori.

Capitolul 11

Mulţimi şi logică fuzzy

11.1 Prezentare generală

Capitolul conţine o introducere a logicii fuzzy şi mulţimilor fuzzy[1]. Domeniile vizează modelarea incertitudinii din lumea reală, raţionamentul aproximativ, imprecizia în exprimare. Majoritatea conceptelor folosite în lumea reală sunt neclare, vagi, ambigue, dar cu toate aceste oamenii operează cu ele foarte bine.

Termenul de "fuzzy" a fost introdus de către Lotfi A. Zadeh, profesor la University of California at Berkley, în lucrarea sa "Fuzzy Sets" [15]. Mulţimile fuzzy – sau mulţimile nuanţate – se bazează pe conceptul de grad de apartenenţă a unui element la o mulţime; acest grad este un număr din intervalul $[0, 1]$, spre deosebire de mulţimile clasice care sunt văzute ca asignând grade de apartenenţă fie 0, fie 1[2]. Într–o mulţime fuzzy, gradul de apartenenţă se exprimă printr-o funcţie cu valori în intervalul $[0, 1]$.

Alături de teoria probabilităţilor, sistemele fuzzy sunt folosite pentru modelarea incertitudinii. Incertitudinea existentă în ceea ce priveşte rezultatul aruncării unui zar este modelată prin variabile aleatoare - urmărindu-se determinarea probabilităţilor asociate diferitelor valori, sau comportamentul obţinut prin repetarea experimentelor etc. Tipul de incertitudine pe care îl abordează sistemele fuzzy este însă diferit. De exemplu, propoziţia "Maria este destul de înaltă" nu are o incertitudine de tip statistic în ea: nu este vorba de evenimente aleatoare repetate sau condiţionări probabiliste. Caracterul vag al unui sistem este o caracteristică intrinsecă a sa; ea nu este dată în vreun fel de observaţii repetate sau încrederea în legătura dintre o stare cunoscută şi una posibil influenţată de ea.

[1]Fuzzy: vag, neclar; în acest context este tradus în limba română şi ca "nuanţat".

[2]Se poate face o paralelă cu funcţia caracteristică definită pentru o submulţime A a lui X:

$$f_A(x) = \begin{cases} 1 & \text{dacă } x \in A \\ 0 & \text{dacă } x \in X \setminus A \end{cases}$$

Insistând pe direcţia aceasta, putem afirma că modul în care se enunţă regulile de producţie bazate pe logica tradiţională:

```
Daca A atunci B
```

este aplicabil doar pentru cazul în care caracterul vag lipseşte cu desăvârşire, de exemplu în matematică. Totuşi, considerând regula: “Dacă e înnorat, atunci va ploua” realizăm că enunţul este vag, cel puţin din cauza următoare: noţiunea de înnorat este vagă – rareori cerul este în totalitate acoperit de nori; vorbim de “parţial înnorat” sau “un pic înnorat” sau “foarte înnorat” şi niciunul din aceşti termeni nu are o caracterizare clară; dacă nu luăm în considerare aceste nuanţe, atunci regula anterioară ar fi utilizabilă doar pentru cazul în care cerul e complet acoperit de nori. Chiar şi “ploaia” poate fi nuanţată – picură, plouă torenţial etc.

Logica fuzzy este asociată deci cu incertitudinea nestatistică. Trăsătura esenţială a teoriei mulţimilor şi a logicii fuzzy este manipularea riguroasă a incertitudinii. Se pun la dispoziţie modalităţi de definire, descriere şi analiză a caracteristicilor vagi.

11.2 Teoria mulţimilor fuzzy

În teoria clasică a mulţimilor, un element fie face parte dintr-o mulţime, fie nu. În mulţimile fuzzy însă, apartenenţa la o mulţime se exprimă printr–un grad de apartenenţă, pentru care valoarea este un număr cuprins între 0 şi 1. Putem vedea aceasta ca o generalizare a mulţimilor clasice: dacă un element aparţine unei mulţimi clasice, atunci valoarea funcţiei de apartenenţă este 1, altfel 0 - de fapt, funcţia caracteristică a unei mulţimi.

Să considerăm de exemplu mulţimea oamenilor înalţi. Evident, putem spune că o persoană care are înălţimea de 2.10 metri face parte din această mulţime. La fel se poate spune şi despre un om cu înălţimea de 2 m sau de 1.90 m; putem nuanţa aici faptele, spunând că ultimele două persoane aparţin într-o măsură mai mică acestei mulţimi. O persoană de 1.60 m sau mai puţin nu mai poate fi considerată ca făcând parte din mulţimea oamenilor înalţi. Soluţia schiţată aici este asignarea unor grade de apartenenţă la o mulţime pentru elementele în discuţie. Să considerăm tabelul 11.1 în care pentru diferite exemple de înălţimi vom specifica gradul de apartenenţă la mulţimea considerată. Mulţimea oamenilor înalţi este considerată din acest moment o mulţime fuzzy (nuanţată).

Observăm că o mulţime fuzzy se poate specifica prin perechi de elemente de grad de apartenenţă/element. O notaţie mai compactă pentru tabelul 11.1 este:

$$\text{Înalt} = \{1/2.10, 0.8/2, 0.6/1.90, 0.4/1.80, 0.2/1.70, 0/1.60\}$$

Persoană	Înălţime	Grad de apartenenţă
A	2.10 m	1
B	2 m	0.8
C	1.90 m	0.6
D	1.80 m	0.4
E	1.70 m	0.2
F	1.60 m	0.0

Tabela 11.1: Înălţimi şi gradul de apartenenţă la mulţimea fuzzy a oamenilor înalţi.

Valorile extreme 0 şi 1 se pot interpreta astfel: dacă $\mu_A(x) = 1$ atunci spunem că x cu certitudine aparţine lui A, dacă $\mu_A(x) = 0$ atunci x cu certitudine nu aparţine lui A.

O altă variantă este specificarea unei funcţii de apartenenţă $\mu_A(x)$, unde μ_A este o funcţie cu valori în intervalul $[0, 1]$, A este o mulţime fuzzy, x este un element din universul discursului pentru care se stabileşte gradul de apartenenţă la mulţimea A. Astfel, $\mu_{\text{Înalt}}(2.10 \text{ m}) = 1$, $\mu_{\text{Înalt}}(1.90 \text{ m}) = 0.6$ etc.

Funcţiile de apartenenţă se pot reprezenta grafic, pe axa orizontală fiind valori al elementelor, iar pe verticală valoarea funcţiei de apartenenţă, precum în figurile 11.1 sau 11.2.

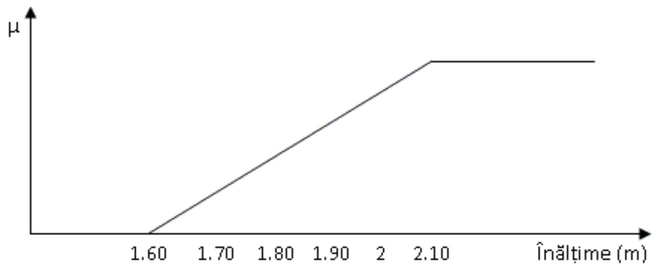


Figura 11.1: Reprezentarea grafică a funcţiei de apartenenţă pentru mulţimea "înalt"

Formele poligonale date în graficele din figura 11.1 şi 11.2 nu sunt singurele care se pot folosi. Se poate de exemplu utiliza o funcţie de tip Gaussian pentru modelarea gradului de apartenenţă:

$$\mu_{\text{Cald}}(T) = e^{-\frac{(T-25)^2}{50}}$$

unde T este temperatura exprimată în grade Celsius. Totuşi, funcţiile formate cu porţiuni liniare sunt mai uşor de calculat şi în practică au un comportament bun; eventuala lor nederivabilitate nu este o problemă. Din raţiuni evidente, spunem că funcţia din figura 11.2 este triunghiulară.

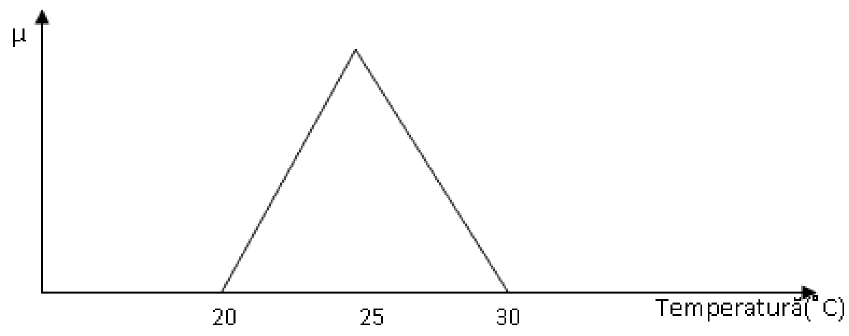


Figura 11.2: Reprezentarea grafică a funcţiei de apartenenţă pentru mulţimea “cald”

În sistemele fuzzy un element poate să aparţină la două mulţimi fuzzy, simultan. De exemplu, o persoană cu înălţimea de 1.75 metri face parte din mulţimea oamenilor înalţi în măsura 0.3[3] şi totodată aparţine mulţimii oamenilor de înălţime medie în măsura 0.45 - a se vedea graficele din figura 11.3. Remarcăm că noţiunile nu se exclud reciproc.

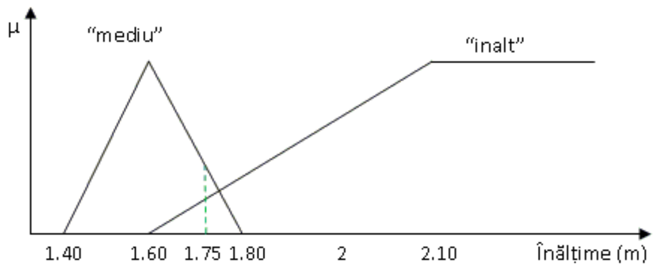


Figura 11.3: Valorile fuzzy “Mediu” şi “Înalt” reprezentate pe acelaşi grafic

11.3 Operaţii cu mulţimi fuzzy

Raţionamentul presupune operaţii logice; o mare parte din noţiunile şi operaţiile din algebra booleană au fost preluate şi adaptate la logica fuzzy.

Înainte de a defini aceste operaţii, este cazul să enumerăm două paradoxuri din logica binară:

1. (Paradoxul mincinosului, paradoxul cretanului) O persoană spune: “eu mint”. Dacă presupunem că această propoziţie este adevărată, atunci înseamnă că spusele persoanei sunt false, deci de fapt ea nu minte,

[3] Valoarea exactă se află intersectând o dreaptă verticală care trece prin valoarea 1.75 de pe abscisă cu graficul funcţiei de apartenenţă.

ceea ce e o contradicţie cu presupunerea iniţială. Dacă presupunem că afirmaţia persoanei este falsă, atunci înseamnă că dimpotrivă, persoana nu minte, deci din nou contradicţie cu presupunerea noastră. Oricare din cele două valori de adevăr am vrea să asociem afirmaţiei "eu mint", ajungem la o contradicţie. Ori, cum doar una din valorile Adevărat şi Fals pot fi asociate unei propoziţii[4], avem un paradox.

2. (Paradoxul bărbierului) Într–un sat există un bărbier care barbiereşte pe toţi bărbaţii care nu se bărbieresc singuri. Care este valoarea de adevăr a propoziţiei "Bărbierul se bărbiereşte singur"? Printr-un procedeu asemănător cu cel anterior, se ajunge la concluzia că niciuna din cele două valori de adevăr nu pot fi asociate propoziţiei, pentru că s–ar ajunge la contradicţie.

Aceste probleme sunt rezolvate de către logica fuzzy, putându–se da valori de adevăr pentru propoziţiile discutate. Intuitiv, pentru ambele afirmaţii am putea spune că ele sunt tot atât de adevărate pe cât sunt de false.

Vom trece acum la definirea operaţiilor pentru mulţimi fuzzy. Definiţiile îi aparţin lui Zadeh.

11.3.1 Egalitatea mulţimilor fuzzy

În teoria clasică a mulţimilor, două mulţimi sunt egale dacă au aceleaşi elemente. Pentru că o mulţime fuzzy înseamnă elemente cu grad de apartenenţă la ea, spunem că două mulţimi fuzzy sunt egale dacă pentru domenii de valori identice au exact aceleaşi valori ale funcţiilor de apartenenţă.

11.3.2 Incluziunea mulţimilor fuzzy

În teoria clasică a mulţimilor, o mulţime A este o submulţime a mulţimii B dacă orice element din A se găseşte şi în B. Pentru cazul mulţimilor fuzzy, folosim următorul exemplu ca suport intuitiv pentru definirea incluziunii: mulţimea oamenilor foarte înalţi este inclusă în mulţimea oamenilor înalţi. Evident, pentru un element x pentru care $\mu_{\text{foarte înalt}}(x) = m$, valoarea asociată lui faţă de de mulţimea oamenilor înalţi este cel puţin la fel de mare: $\mu_{\text{înalt}}(x) = m + \varepsilon, \varepsilon \geq 0$. Ca atare, spunem că mulţimea fuzzy A este inclusă în mulţimea fuzzy B dacă cele două mulţimi conţin aceleaşi elemente şi $\mu_A(x) \leq \mu_B(x)$, $\forall x$. Desigur, şi alte definiţii sunt posibile.

11.3.3 Complementara unei mulţimi fuzzy

În teoria clasică a mulţimilor, complementul unei mulţimi A este mulţimea formată din toate elementele care nu aparţin lui A.

[4]Conform principiului terţului exclus, a treia variantă nu este posibilă.

Pentru definiţia relativ la mulţimi fuzzy, pornim de la un exemplu: considerăm mulţimea oamenilor de înălţime medie, pentru care funcţia de apartenenţă este dată în figura 11.3. Se pune problema: când spunem că o persoană nu este de înălţime medie? Dacă persoana are înălţimea 1.75 m, evident că face parte din mulţimea oamenilor de înălţime medie; dacă are 1.90 m sau 1.80 m, atunci e evident că nu face parte din ea. Pentru o persoană pentru care gradul de apartenenţă la mulţimea oamenilor de înălţime medie este, să spunem, 0.3, pare rezonabil să spunem că ea nu aparţine la această mulţime cu măsura $1 - 0.3 = 0.7$. Putem deci defini valoarea de apartenenţă a complementului mulţimii ca fiind unu minus valoarea de apartenenţă la mulţime. Desigur, şi alte definiţii sunt posibile.

Definiţia dată contrazice principiul terţului exclus: in logica clasică se spune că ceva fie este A, fie este non-A. Gradul de adevăr pentru afirmaţiile discutate în cele două paradoxuri poate fi luat 0.5, deci fiecare propoziţie are aceeaşi valoare de adevăr ca şi contrara ei.

11.3.4 Intersecţia a două mulţimi fuzzy

În varianta clasică, intersecţia a două mulţimi este o mulţime formată din elementele comune celor care se intersectează.

Definirea pentru mulţimi fuzzy nu este unică; oricare ar fi varianta folosită, trebuie să se respecte următoarele:

1. operaţia să fie comutativă: $\mu_{A\cap B}(x) = \mu_{B\cap A}(x)$
2. de asemenea, să avem asociativitate: $\mu_{(A\cap B)\cap C}(x) = \mu_{A\cap(B\cap C)}(x)$
3. monotonie: dacă valoarea funcţiei de apartenenţă a unui element la o mulţime scade, atunci valoarea funcţiei de apartenenţă pentru mulţimea respectivă intersectată cu o alta nu trebuie să crească.

În practică, cel mai mic grad de apartenenţă relativ la cele două mulţimi determină gradul de apartenenţă la intersecţie. Varianta de operator de intersecţie dată de către Zadeh este:

$$\mu_{A\cap B}(x) = \min\{\mu_A(x), \mu_B(x)\}$$

Se poate arăta uşor că dacă $A \subset B$, atunci $A \cap B = A$, în sens fuzzy, ceea ce este în concordanţă cu ceea ce avem şi în teoria clasică a mulţimilor. Desigur, şi alte definiţii sunt posibile pentru acest operator.

11.3.5 Reuniunea a două mulţimi fuzzy

În teoria clasică a mulţimilor, reuniunea a două mulţimi este o mulţime formată din toate elementele care se regăsesc în ele. Pentru definirea relativ la mulţimi fuzzy, se iau în considerare proprietăţi similare cu cele de la intersecţie (doar la monotonie apare diferenţa), iar varianta dată de către Zadeh este:

$$\mu_{A\cup B}(x) = \max\{\mu_A(x), \mu_B(x)\}$$

Se pot da şi alte definiţii pentru acest operator.

11.3.6 Operatori de compensare

În timp ce operaţiile din cadrul teoriei clasice a mulţimilor sunt unic definite, pentru mulţimile fuzzy există şi alte posibilităţi de definire a lor decât cele date mai sus. Operatorii de compensare tratează în special cazul reuniunii şi al interseţiei de mulţimi fuzzy. Intersecţia este des întâlnită în cadrul regulilor fuzzy.

Folosind definiţia intersecţiei dată de Zadeh, valoare funcţiei de apartenenţă pentru intersecţia a 2 mulţimi fuzzy este controlată de cea mai mică din valorile existente. De exemplu, pentru regula "dacă A şi B şi C atunci D", dacă valorile de apartenenţă ale lui A, B şi C sunt respectiv 0.2. 0.8, 0.9, efectul pe care îl are A asupra rezultatului final este prea pronunţat. În practică, definiţia intersecţiei dată de Zadeh nu e întotdeauna adecvată.

S–au definit mai mulţi operatori de compensare. Ei formulează răspunsuri la întrebarea: cât de mult poate să compenseze creşterea unor variabile valorile mici ale altora? Vom prezenta două variante ale acestor operatori: operatorul de medie şi operatorul gama.

Prin operatorul de medie se stabileşte că valoarea funcţiei de apartenenţă este media valorilor individuale:

$$\mu_{X_1\cap X_2\cap\cdots\cap X_n}(x) = \frac{\sum\limits_{i=1}^{n}\mu_{X_i}(x)}{n}$$

Operatorul gama este mai complex şi pare să reprezinte mai bine procesul de decizie umană decât definiţiile lui Zadeh. El este definit ca:

$$\mu_\gamma = \left(\prod_{i=1}^{n}\mu_i\right)^{1-\gamma}\cdot\left(1-\prod_{i=1}^{n}(1-\mu_i)\right)^{\gamma}$$

unde $0 \leq \gamma \leq 1$, iar n este numărul de valori fuzzy implicate în intersecţie. În practică, cel mai frecvent valorile lui γ sunt între 0.2 şi 0.4.

11.4 Reguli fuzzy

Regulile clasice au forma următoare:

Dacă A_1 şi A_2 şi $\cdots$ şi A_n atunci C

unde "A_1 şi A_2 şi $\cdots$ şi A_n" se numeşte antecedent sau premisă iar C este consecvent sau consecinţă sau concluzie. De exemplu:

Dacă înălţimea bărbatului este mai mare de 1.80 m, atunci masa lui este mai mare de 50 kg.

Regulile fuzzy păstrează această formă generală, dar pot să apară diferenţe pe partea de consecvent. Cele două variante des folosite sunt datorate lui Mamdani:
Dacă X_1 *este* A_1 *şi* $\ldots$ *şi* X_n *este* A_n *atunci* Y *este* B
şi respectiv lui Takagi, Sugeno şi Kang:
Dacă X_1 *este* A_1 *şi* $\ldots$ *şi* X_n *este* A_n *atunci* $Y = p_0 + p_1X_1 + \cdots + p_nX_n$
unde X_i sunt variabile fuzzy de intrare, A_i sunt mulţimi fuzzy peste variabilele X_i $(1 \leq i \leq n)$, Y este o variabilă fuzzy de ieşire, B este o mulţime fuzzy definită peste valorile lui Y iar p_j sunt coeficienţi reali $(0 \leq j \leq n)$.

Pentru probleme de clasificare există următoarea formă de regulă:
Dacă X_1 *este* A_1 *şi* $\ldots$ *şi* X_n *este* A_n *atunci* Y *face parte din clasa i în măsura* GC_i.

Nuanţarea[5] este pasul prin care se combină valorile din antecedentul unei reguli, folosind operaţiile cu mulţimi fuzzy; pasul se aplică pentru fiecare regulă în parte. Prin combinarea regulilor date la pas de denuanţare se obţine o ieşire care poate fi folosită ca rezultat inferenţial sau ca indicaţie de control al unui sistem.

Exemplificarea acestor reguli se face pentru cazul unei centrale de încălzire, pentru care sunt date nişte reguli privind reglarea debitului de gaz astfel încât să se obţină o temperatură potrivită. Se pleacă de la reguli în care se folosesc noţiuni vagi (temperatură potrivită, variaţie mare, măreşte debitul etc.) şi se obţine o indicaţie pentru regulatorul de gaz.

Vom considera că avem valori de intrare precum temperatura interioară (notată $TempIn$), cea exterioară ($TempExt$), modificarea de temperatură interioară în ultimele 5 minute ($DeltaTempIn$); ca valoare de ieşire avem $ModificareDebit$. Fiecare valoare de intrare concretă va avea un grad de apartenenţă fuzzy la diferite mulţimi (de exemplu, pentru $TempIn$ avem apartenenţă la mulţimi precum *rece*, *confortabil* etc.

Pentru valoarea $TempIn$ avem trei mulţimi fuzzy: *rece*, *confortabil*, *prea cald*. Pentru $TempExt$ avem mulţimile fuzzy *foarte rece*, *rece*, *cald*, *foarte cald* şi *fierbinte*. Pentru $DeltaTempIn$ definim mulţimile fuzzy: *larg negativ*, *mic negativ*, *aproximativ zero*, *pozitiv mic*, *larg pozitiv*[6], iar pentru $ModificareDebit$ avem seturile fuzzy *scade mult*, *scade puţin*, *nu schimba*, *creşte puţin*, *creşte mult*.

Vom considera doar câteva reguli, suficiente pentru exemplificarea nuanţării şi denuanţării:

Regula 1: Dacă $TempIn$ este *confortabilă* şi $DeltaTempIn$ este *aproximativ zero*, atunci $ModificareDebit$ este *nu schimba*;

[5] În original: fuzzyfication.
[6] "Mic" şi "larg" se referă la valorile absolute (modulul) cantităţilor măsurate.

Regula 2: Dacă $TempExt$ este *rece* şi $DeltaTempIn$ este *mic negativ*, atunci $ModificareDebit$ este *creşte puţin*;

Regula 3: Dacă $TempIn$ este *prea cald* şi $DeltaTempIn$ este *larg pozitivă*, atunci $ModificareDebit$ este *scade mult*;

Regula 4: Dacă $TempIn$ este *rece* şi $DeltaTempIn$ este *aproximativ zero*, atunci $ModificareDebit$ este *creşte puţin*.

Pentru $TempIn$, mulţimea *confortabil* se defineşte ca $\{0/15°, 1/21°, 0/27°\}$, interpretată ca o funcţie de apartenenţă de tip triunghiular. De exemplu, pentru 18° şi 24° gradele de apartenenţă sunt ambele 0.5. Pentru *rece* avem mulţimea fuzzy $\{1/10°, 1/16°, 0/21°\}$, iar *prea cald* este mulţimea $\{0/21°, 1/27°, 1/33°\}$.

Pentru $DeltaTempIn$:

$$\begin{aligned} negativ\ mic &= \{0/-4°, 1/-2°, 0/0°\} \\ aproape\ zero &= \{0/-2°, 1/0°, 0/+2°\} \\ larg\ pozitiv &= \{0/2°, 1/4°, 1/6°\} \end{aligned}$$

Pentru $TempExt$, *rece*=$\{0/-1°, 1/10°, 0/21°\}$.

Să presupunem că temperatura interioară este de 20°, diferenţa de temperatură din ultimele 5 minute este $-1.5°$ iar temperatura exterioară este de 11°.

Conform mulţimilor fuzzy date anterior, avem:

- pentru $TempIn$, $\mu_{rece}(20°) = 0.25$, $\mu_{confortabil}(20°) = 0.75$, $\mu_{prea\ cald}(20°) = 0$;
- pentru $DeltaTempIn$, $\mu_{mic\ negativ}(-1.5°) = 0.80$, $\mu_{aproximativ\ zero}(-1.5°) = 0.20$, $\mu_{larg\ pozitiv}(-1.5°) = 0$
- pentru $TempExt$, $\mu_{rece}(11°) = 0.90$

Aplicăm aceste valori celor 4 reguli fuzzy de mai sus. Ţinem cont de faptul că antecedentele din reguli sunt exprimate cu conjuncţie, corespunzătoare interseţiei de mulţimi, ceea ce în logica fuzzy se implementează prin funcţia min. Obţinem:

Regula 1: $0.75 \cap 0.20 = 0.20 = \mu_{nu\ schimba}(ModificareDebit)$

Regula 2: $0.90 \cap 0.80 = 0.80 = \mu_{creste\ putin}(ModificareDebit)$

Regula 3: $0 \cap 0 = 0 = \mu_{scade\ mult}(ModificareDebit)$

Regula 4: $0.25 \cap 0.20 = 0.20 = \mu_{creste\ putin}(ModificareDebit)$

Activarea acestor reguli se face în paralel. Observăm că pentru regula a treia ieşirea este zero, deci ea nu se va aplica. Din regulile 2 şi 4 avem două valori pentru apartenenţa $\mu_{creste\ putin}(ModificareDebit)$; se va lua maximul celor două valori, deci $\mu_{creste\ putin}(ModificareDebit) = 0.8$.

Variaţia de debit este modelată la rândul ei fuzzy, aşa cum se arată în figura 11.4.

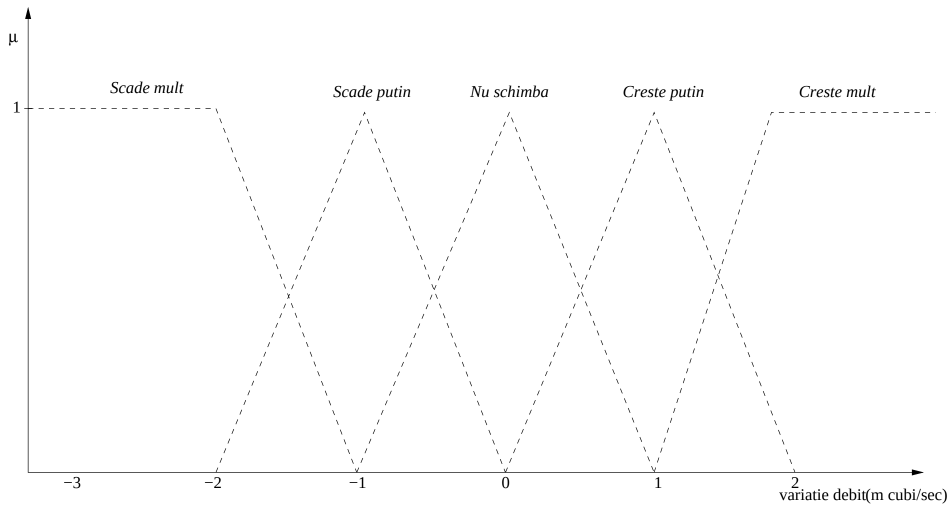


Figura 11.4: Mulţimi fuzzy pentru debitul de gaz

Denuanţarea este operaţia prin care se obţine un răspuns concret la problemă, adică se furnizează o valoare exprimată în metri cubi pe secundă pentru debitul de gaz. Plecând de la graficul anterior, se trasează conturul mărginit de orizontalele $y = 0.2$ - pentru *nu schimba* - şi $y = 0.8$ - pentru *creşte puţin*. Se obţine figura geometrică desenată cu linie continuă în figura 11.5, pentru care se determină centrul de greutate; verticala dusă prin acest centru de greutate interesectează axa debitului la valoarea +0.76, aceasta fiind indicaţia dată regulatorului de gaz: creşte cu 0.76 m^3/s debitul de gaz.

Există mai multe metode care se pot folosi pentru denuanţare; a se vedea [5] pentru detalii.

11.5 Măsuri ale gradului de nuanţare

În cazul unei mulţimi fuzzy se poate pune întrebarea: cât este ea de nuanţată? Pentru o mulţime fuzzy discretă, se pot introduce câteva măsuri care cuantifică gradul de nuanţare. Acestea au ca scop măsurarea gradului de incertitudine, care apare, de exemplu, în cazul exprimării vagi. În continuare, prezentarea merge pe exemplele din [5].

Dacă considerăm mulţimea peştilor, atunci pentru diferite elemente de mai jos avem gradele exprimate: $\mu_{peşti}(biban) = 1.0$, $\mu_{peşti}(peştisor\ auriu) =$

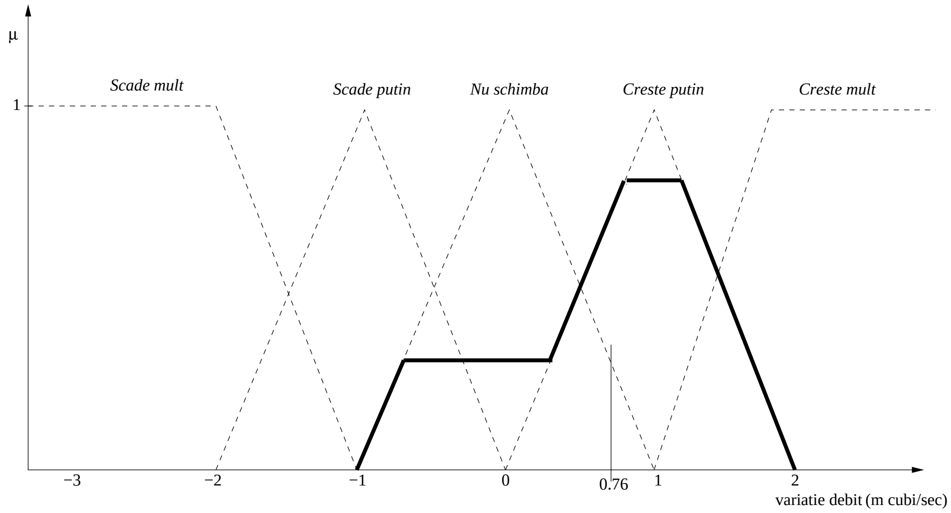


Figura 11.5: Procesul de denuanţare.

1.0, $\mu_{peşti}$(*căluţ de mare*) = 0.8, $\mu_{peşti}$(*balenă*) = 0.0. Pentru mulţimea fuzzy *flori*, gradul de apartenenţă ar putea fi: μ_{flori}(*trandafir*) = 1.0, μ_{flori}(*pâine*) = 0.0, μ_{flori}(*lemn câinesc*) = 0.5. Intuitiv, putem spune că mulţimea de flori exemplificată este mai vagă decât mulţimea peştilor: în ultimul caz, valorile de apartenenţă sunt mai apropiate de cele ale unei funcţii de apartenenţă din cazul unei mulţimi clasice, 0 şi 1.

Se preia din fizică şi din teoria informaţiei noţiunea de *entropie*, care măsoară gradul de dezorganizare a unui sistem. Spre deosebire de teoria informaţiei unde măsura entropiei este unic determinată[7], în teoria mulţimilor fuzzy sunt mai multe variante acceptate. Se pleacă de la o sumă de proprietăţi pe care ar trebui să le respecte o astfel de măsură entropică şi se introduc mai multe funcţii care respectă o parte sau toate aceste proprietăţi. De exemplu, pentru o mulţime clasică, din care un element fie face parte, fie nu, măsura gradului de nuanţare ar trebui să dea rezulatul 0 - *i.e.* o mulţime clasică nu este vagă. De asemenea, cu cât sunt mai multe valori pentru care valoarea funcţiei de apartenenţă este 0.5 sau apropiată de aceasta, cu atât mulţimea este mai vagă: un element care aparţine cu măsura 0.5 la o mulţime fuzzy face şi nu face parte din mulţimea dată în aceeaşi măsură.

Înainte de a da diferite variante de măsurare a gradului de nuanţare, se introduce noţiunea de "mulţime mai ascuţită", ce exprimă relaţia între două mulţimi fuzzy: spunem că o mulţime S^* este *mai ascuţită* decât o altă mulţime fuzzy S – ambele definite peste acelaşi univers al discursului – dacă $\mu_{S^*}(x) \leq \mu_S(x)$ pentru cazul în care $\mu_S(x) < 0.5$ şi $\mu_{S^*}(x) \geq \mu_S(x)$ dacă $\mu_S(x) > 0.5$; pentru $\mu_S(x) = 0.5$ valoarea $\mu_{S^*}(x)$ poate fi oricât.

Proprietăţile de mai jos sunt punct de plecare pentru determinarea dife-

[7]Abstracţie făcând de o constantă multiplicativă

ritelor funcţii de măsurare a gradului de nuanţare.

caracterul exact: $H(A) = 0$ dacă şi numai dacă A este o mulţime clasică;

maximalitatea: $H(A)$ este maximă dacă $\mu_A(x) = 0.5$, $\forall x$ din universul discursului;

ascuţirea: $H(A) \geq H(A^*)$ dacă A^* este mai ascuţită decât A;

simetria: $H(A) = H(\overline{A})$;

principiul includerii şi excluderii: $H(A \cup B) = H(A) + H(B) - H(A \cap B)$

O variantă de funcţie de măsurare a gradului de nuanţare, introdusă de DeLuca şi Termini şi care respectă toate cele 5 proprietăţi de mai sus este:

$$H_{DT}(A) = -K \sum_{i=1}^{n} (\mu_i \log \mu_i + (1 - \mu_i) \log(1 - \mu_i)) \qquad (11.1)$$

unde K este un număr pozitiv oarecare. Varianta introdusă de Pal şi Pal este:

$$H_{PP}(A) = K \sum_{i=1}^{n} \left(\mu_i e^{1-\mu_i} + (1 - \mu_i) e^{\mu_i} \right) \qquad (11.2)$$

Se pot defini şi alte variante de măsurare a gradului de nuanţare a unei mulţimi vagi.

Bibliografie

[1] S. S. Haykin, *Neural networks and learning machines.* Upper Saddle River, NJ: Pearson Education, third ed., 2009.

[2] R. Andonie and A. Cațaron, *Inteligenţă computaţională.* Universitatea Transilvania din Braşov, 2002. http://vega.unitbv.ro/~cataron/Publications/curs_rn.pdf.

[3] J. Zurada, *Introduction to Artificial Neural Systems.* West Publishing Co., 1992.

[4] A. P. Engelbrecht, *Computational Intelligence: An Introduction.* Wiley, 2002.

[5] R. C. Eberhart and Y. Shi, *Computational intelligence - concepts to implementations.* Elsevier, 2007.

[6] "Stanford Machine Learning, curs online Coursera." https://www.coursera.org/learn/machine-learning, 2019. Accesat: 2019-09-30.

[7] K. B. Petersen and M. S. Pedersen, "The matrix cookbook," 2008. Version 20081110.

[8] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning.* MIT Press, 2016. http://www.deeplearningbook.org.

[9] M. A. Nielsen, "Neural networks and deep learning," 2018.

[10] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, ICCV '15, p. 1026–1034, IEEE Computer Society, 2015.

[11] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics* (Y. W. Teh and M. Titterington, eds.), vol. 9 of *Proceedings of Machine Learning Research*, pp. 249–256, PMLR, 13–15 May 2010.

[12] D. Arthur and S. Vassilvitskii, "k-means++: the advantages of careful seeding," in *SODA '07: Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, (Philadelphia, PA, USA), pp. 1027–1035, Society for Industrial and Applied Mathematics, 2007.

[13] G. A. Carpenter, S. Grossberg, N. Markuzon, J. H. Reynolds, and D. B. Rosen, "Fuzzy artmap: A neural network architecture for incremental supervised learning of analog multidimensional maps," *IEEE transactions on neural networks*, vol. 3 5, pp. 698–713, 1992.

[14] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs.* Berlin, Heidelberg: Springer-Verlag, third ed., 1996.

[15] L. Zadeh, "Fuzzy sets," *Information and Control*, vol. 8, no. 3, pp. 338 – 353, 1965.